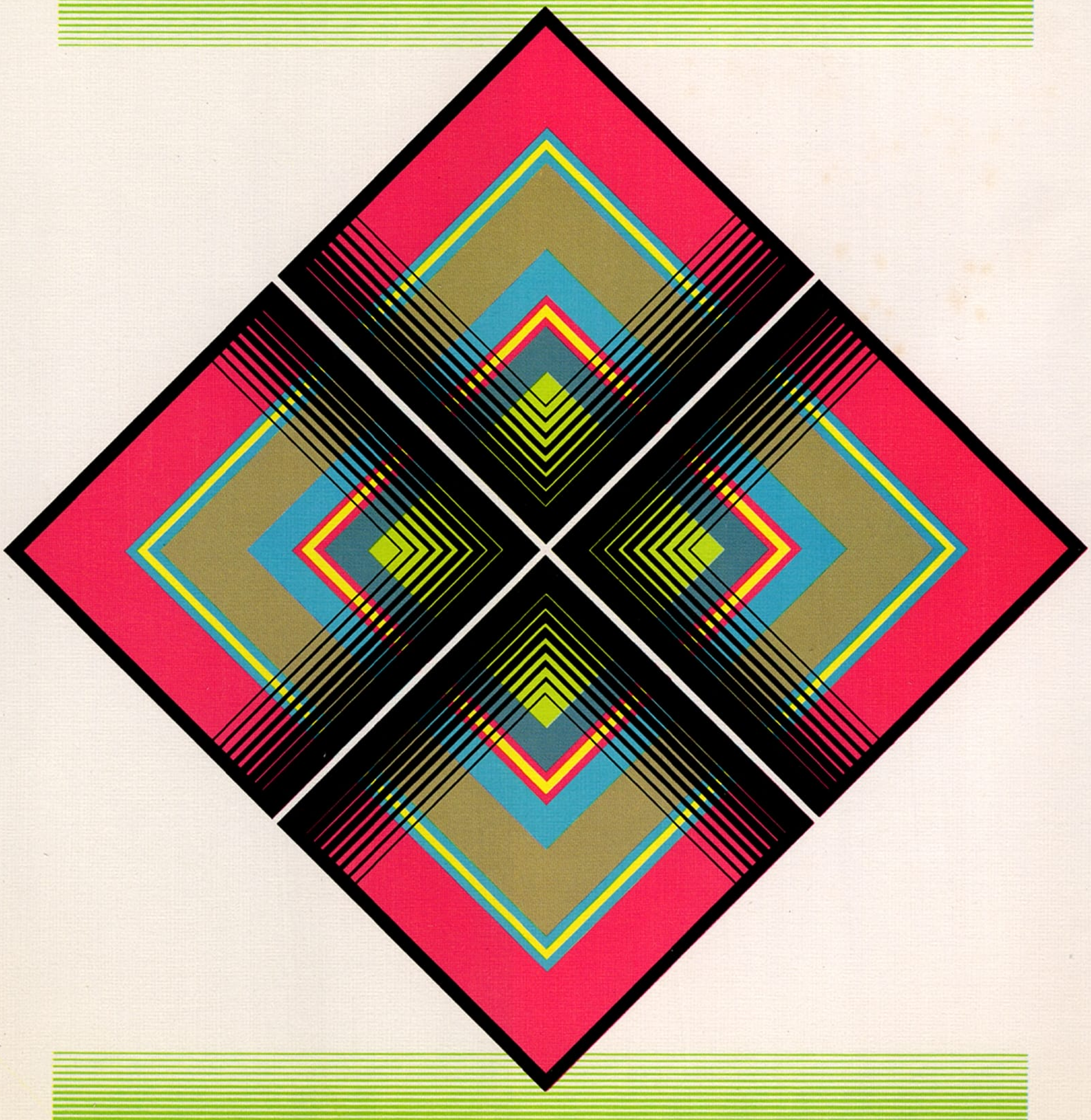
Personal Computer

PC-8801 +mkⅡ

NEC

マシン語プログラミング入門

森巧尚＝著



ナツメ社

Personal Computer
PC-8801＋mkⅡ
NEC
マシン語プログラミング入門
森巧尚＝著
ナツメ社

[本書を読まれる方へ]

⊙「マシン語を使って、こんなことがしてみたい」
この気持です。
この気持こそが、あなたに
マシン語を教えてくれるのです。

⊙パソコンユーザーの中には、
マシン語はむずかしい、
と誤解している人が多いようです。
確かにマシン語には、たくさんの命令群があり、
それらをすべて使いこなすようになるには、
かなりの努力が必要です。
しかし実際に、私たちがやってみたいことを
マシン語でコーディングするとき、
使用するコマンドはそれほど多くはありません。

⊙ベーシックでのプログラミングを
思い浮かべてください。
ひんぱんに使うコマンド、ステートメントは
かなり限定されていましたね。
マシン語についても、まったく同じことがいえます。
少ない命令でかなりのことができますし、
また、プログラミングの考え方もベーシックと同じです。

⊙とにかく、この本でチャレンジしてみてください。
あなたはすでに「やる気」はお持ちなのですから……。

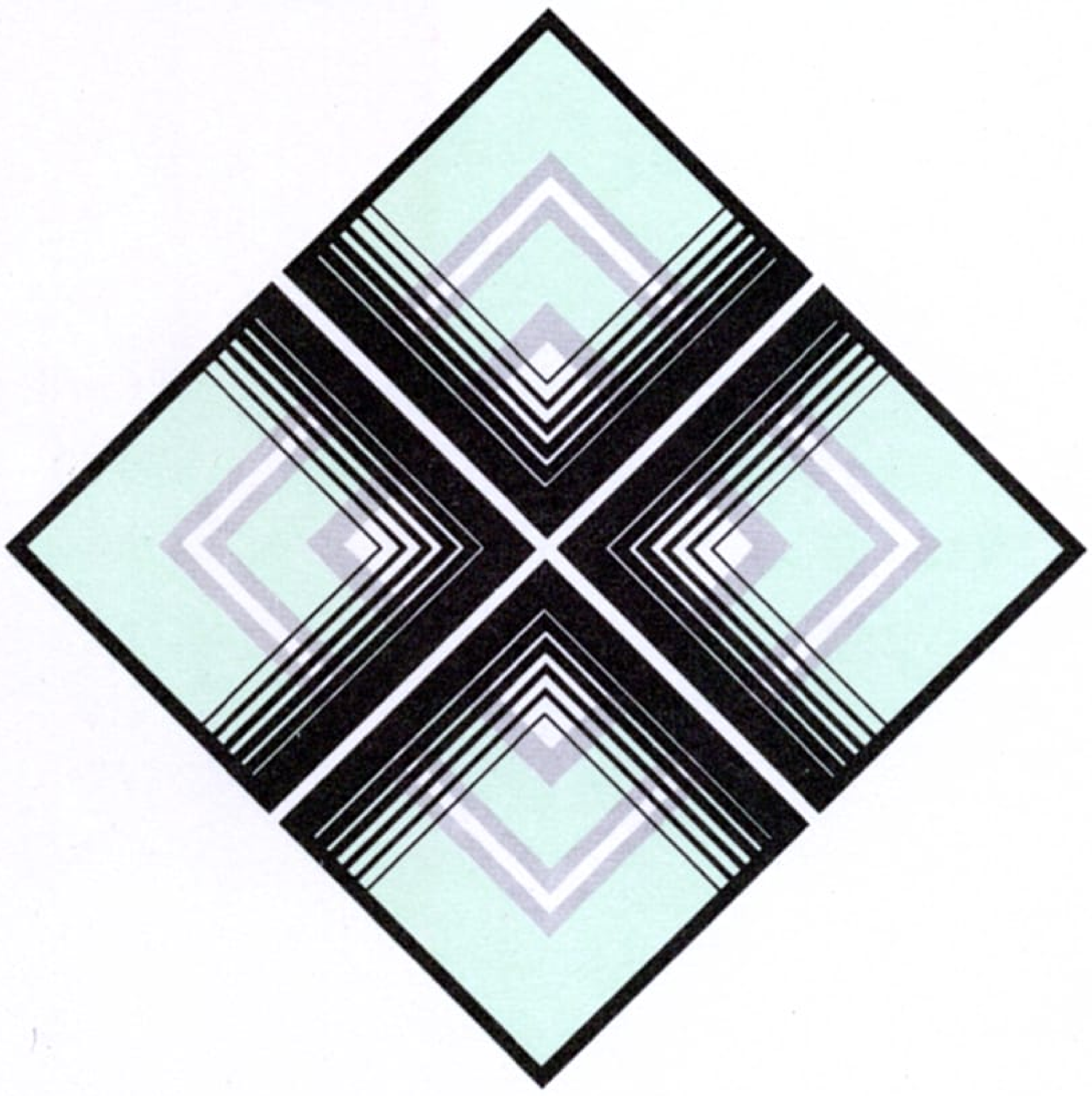
cover design——nobuo nakagaki + yoshifumi hayase

Personal Computer

NEC PC-8801 +mkII

マシン語プログラミング入門

森巧尚＝著



ナツメ社

はじめに

「マシン語でこんなことがしたい」

この気持ちです。
この気持ちこそがあなたにマシン語を教えてくれます。

マシン語とは論理的なもので、いろいろな命令をいかに使いこなすかが重要なものだと思い込んでおられるでしょう。完全にマシン語をマスターしてしまった人の中にもそういう人はいると思います。

確かにすぐれたプログラムを作ろうという時、命令の使い方をたくさん知っていればいるほど、よりすぐれたものになるでしょう。しかし、それはあくまでも表面的なことです。

命令の使い方を少ししか知らなくても、**「こんなプログラムを作りたい」**という気持ちがあればそれで十分です。それが大事なのです。

「こんなプログラム」というイメージがはっきり見えて来れば勝手にプログラムは出来上がってしまいます。途中でわからなくなっても、ちょっと本を読むだけでスラスラ理解できますし、そうでなくても直観でわかってしまいます。本はあくまでもヒントにすぎません。

アイデアや創造性がないからプログラムが作れない。全然違います。その前に「心」が必要なのです。どうしても作りたくて作りたくてしかたがないから、アイデアが湧き、創造できるのです。

論理的なこと、これはすべてコンピュータにまかせてしまって下さい。あなたは、何がしたいか、どうやりたいか、感性を磨くのです。

1984年　6月27日

フォーサイト企画部　森　巧尚

もくじ

第1章

Are you ready?

1・1 THE GALAXY !!

それこそ星の数ほどもある無数の美しい星たち。
さあ！ 星の世界へトリップしましょう！

使い方

〈注〉本節でのサンプルプログラムは「ベーシックプログラム」と「マシン語プログラム」から構成されていますが、入力方法がわからない方も多いと思います。そのような方は、本節を飛ばして読み進み、あとでここへ戻って入力実行してみてください。なお、右ページのプログラムは「ディスクベース」で走ります。

プログラムを走らせると、そこはもう星の世界！

キラキラとまたたきながら落ちてゆく美しい世界に、しばらく見とれていることにいたしましょう。

でも、もっと感激してみたい人は、もっともっと感激することができるのです。

テンキーの [1] と [3] を交互に押してみて下さい。

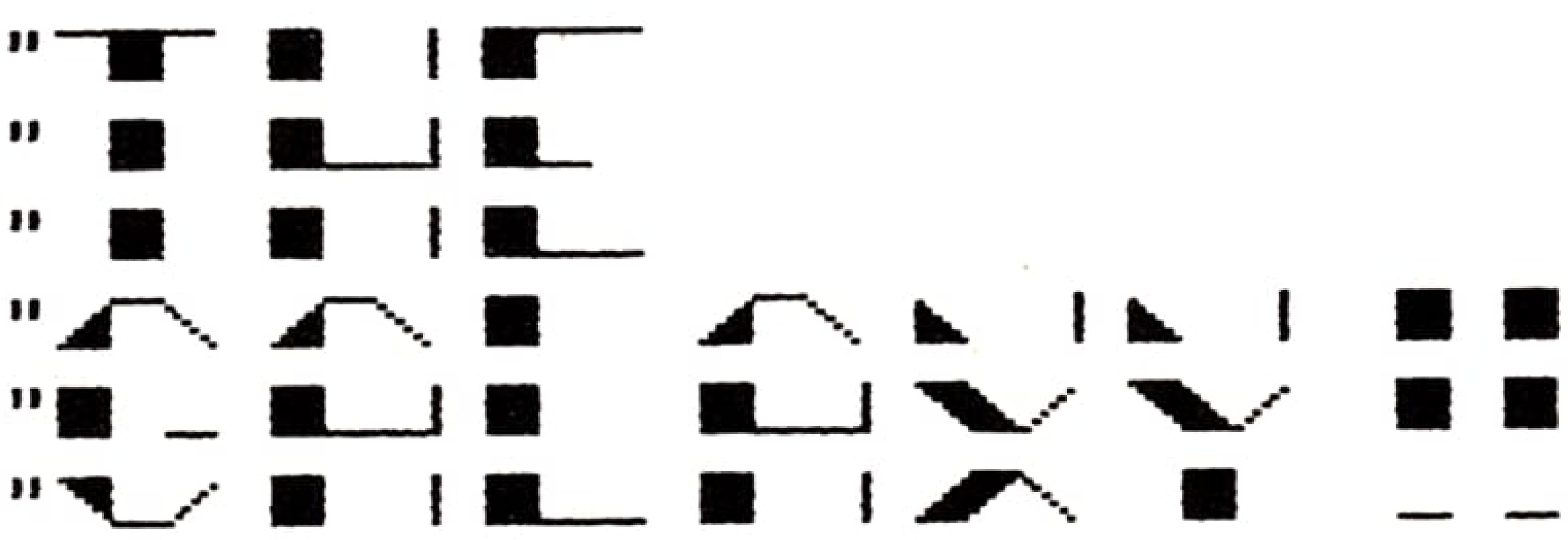
いかがですか？ 加速度をつけて左右へ流れていく星々を見送っていくうちに、いつしか気分はＳｐａｃｅ Ｄｒｅａｍｅｒ……。



〈写真１〉 まるで本物のようです。

リスト1——BASICプログラムリスト

```
100 DEFINT A-Z  :DEF USR=&HB980
110 CONSOLE 0,25,0,1 :WIDTH 40,25
120 CLS 3 :PRINT "Wait a moment."
130 '
140 FOR I=0 TO 199
150  POKE &HC000+I*3,INT(RND*&H3F+&HC0)
160  POKE &HC001+I*3,INT(RND*&HFF)
170  POKE &HC002+I*3,INT(RND*7+1)
180 NEXT
190 '
200 CLS :COLOR 5
210 LOCATE  2, 1:PRINT "
220 LOCATE  2, 2:PRINT "
230 LOCATE  2, 3:PRINT "
240 LOCATE 10, 5:PRINT "
250 LOCATE 10, 6:PRINT "
260 LOCATE 10, 7:PRINT "
```



```
270 '
280 N=5
290 I=INP(0)
300 IF I=253 AND N>1  THEN N=N-1
310 IF I=247 AND N<10 THEN N=N+1
320 A=USR(N)
330 GOTO 290
```

リスト2——マシン語プログラムリスト

```
B980 7E C6 EB 32 B7 B9 3A 0C BA FE 00 20 12 3E 03 32
B990 0C BA 3A 0B BA 3D 20 02 3E 04 32 0B BA 18 04 3D
B9A0 32 0C BA 06 C8 11 00 C0 F3 1A 67 13 1A 6F 1B 0E
B9B0 07 AF CD F1 B9 D5 11 F0 00 19 7C 0E 00 B9 20 02
B9C0 26 C0 D1 7C 12 13 7D 12 13 1A 4F 3E 03 CD D5 B9
B9D0 13 10 D6 FB C9 F5 3A 0B BA 3D 28 08 3D 28 11 3D
B9E0 28 09 18 0C 78 FE 96 30 20 18 05 78 FE 32 38 19
B9F0 F1 CB 39 30 03 D3 5C 77 CB 39 30 03 D3 5D 77 CB
BA00 39 30 03 D3 5E 77 D3 5F C9 F1 C9 03 02 00 00 FF
```

1・2　マシン語の世界へ

〈注〉第１章から第５章まではカセットテープ使用（テープベース）を前提に話を進めます。ディスクの使用はできません。ＰＣ－8801mkIIのディスク内蔵タイプをご使用の方は、ディップスイッチＳＷ－２の７番をＯＦＦ（ＲＯＭベースまたはテープベース）にしてください。

いまのプログラムのように私たちは、マイコン雑誌などでよくマシン語を見かけています。

２ケタのほんの少しアルファベットのまじった数字が、ずらっと、目がおかしくなるほど並んでいるもの。きっとそれを連想したことでしょう。

確かにそれがマシン語です。

ＢＡＳＩＣとくらべると、はるかに理解しがたいもののように思えます。

それなのにマシン語にひかれてゆきます。

なぜでしょうか？

外見ではなく、マシン語のすばらしさ自体にひかれているのです。

ＢＡＳＩＣだけでできたプログラムより
ＢＡＳＩＣ＋マシン語でできたプログラム。
ＢＡＳＩＣ＋マシン語でできたプログラムより
マシン語だけでできたプログラム。

私たちは、マシン語は、ＢＡＳＩＣではできないことでもできることを知っています。

マシン語でできないことはないとさえ思っていたりもします。

本当にそうなのでしょうか？

だいたいは、正解と言っていいでしょう。しかし、それは、本当はＢＡＳＩＣとマシン語の速度の違いがひき起こした結果なのです。

ＢＡＳＩＣでもマシン語でもやっていることはほとんど変わりません。

１つの計算をしている時に同時に他の計算をしたりなんてことは、いくらマシン語だからといってもできないことなのです。

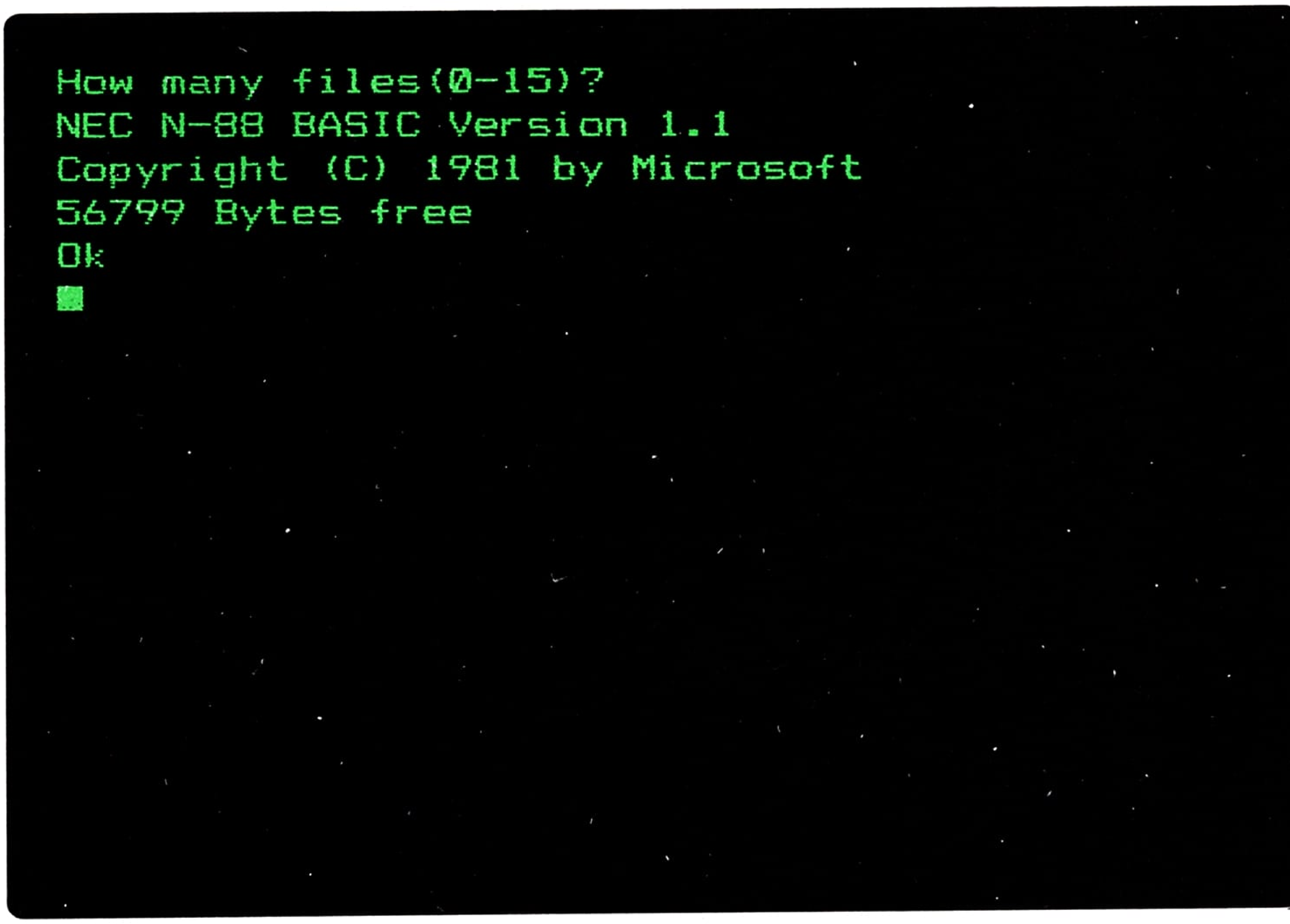
ただ、**その実行速度があまりにもケタはずれに速いため、たくさんのことを同時に行っているように見えている**のです。

さあ、それでは、このすばらしきマシン語の世界へ私たちも入って行くことにいたしましょう！

MON

まずは、本体の電源をＯＮにします。
おなじみの画面があらわれました。

〈注〉下の画面は、前ページ欄外で説明したようにＲＯＭベースでの初期画面です。



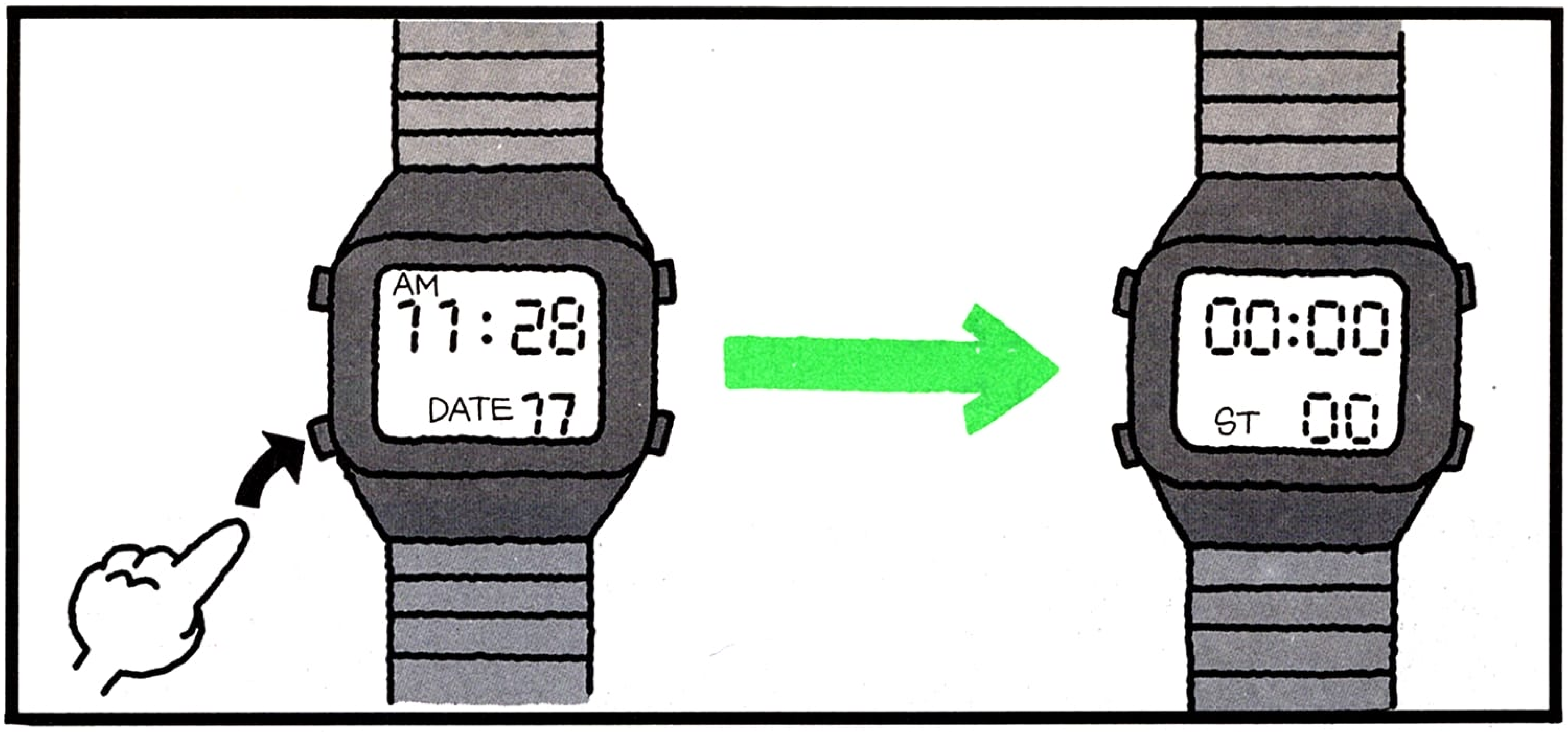
しかし、このままでは**ＢＡＳＩＣのプログラムしか扱えません。**
それは、現在ＢＡＳＩＣを扱う状態になっているからです。

たとえば、デジタル時計の場合は、ふだんは時刻を表示するモードになっています。しかし、ストップウォッチを使いたい時、そのままでは使えません。

なぜなら、今は時刻を表示する回路が働いていて、ストップウォッチの回路は休んでいるからなのです。

つまり、ボタンを押して、ストップウォッチのモードに切りかえないといけないのです。

マイコンの場合も同じだと考えてもらえばよいでしょう。



マシン語を扱うときには、今までの**BASICを扱うモードから**、**マシン語を扱うモードに切りかえる**必要があるのです。

そして、そのマシン語モードに切りかえる命令というのが、

MON

なのです。BASICのモードから、

```
MON ↵
```

または、

```
mon ↵
```

と打ち込みます。

```
How many files(0-15)?
NEC N-88 BASIC Version 1.1
Copyright (C) 1981 by Microsoft
56799 Bytes free
Ok
mon

h]■
```

さあ、これでマシン語を自由に扱うことができます。

CTRL + B

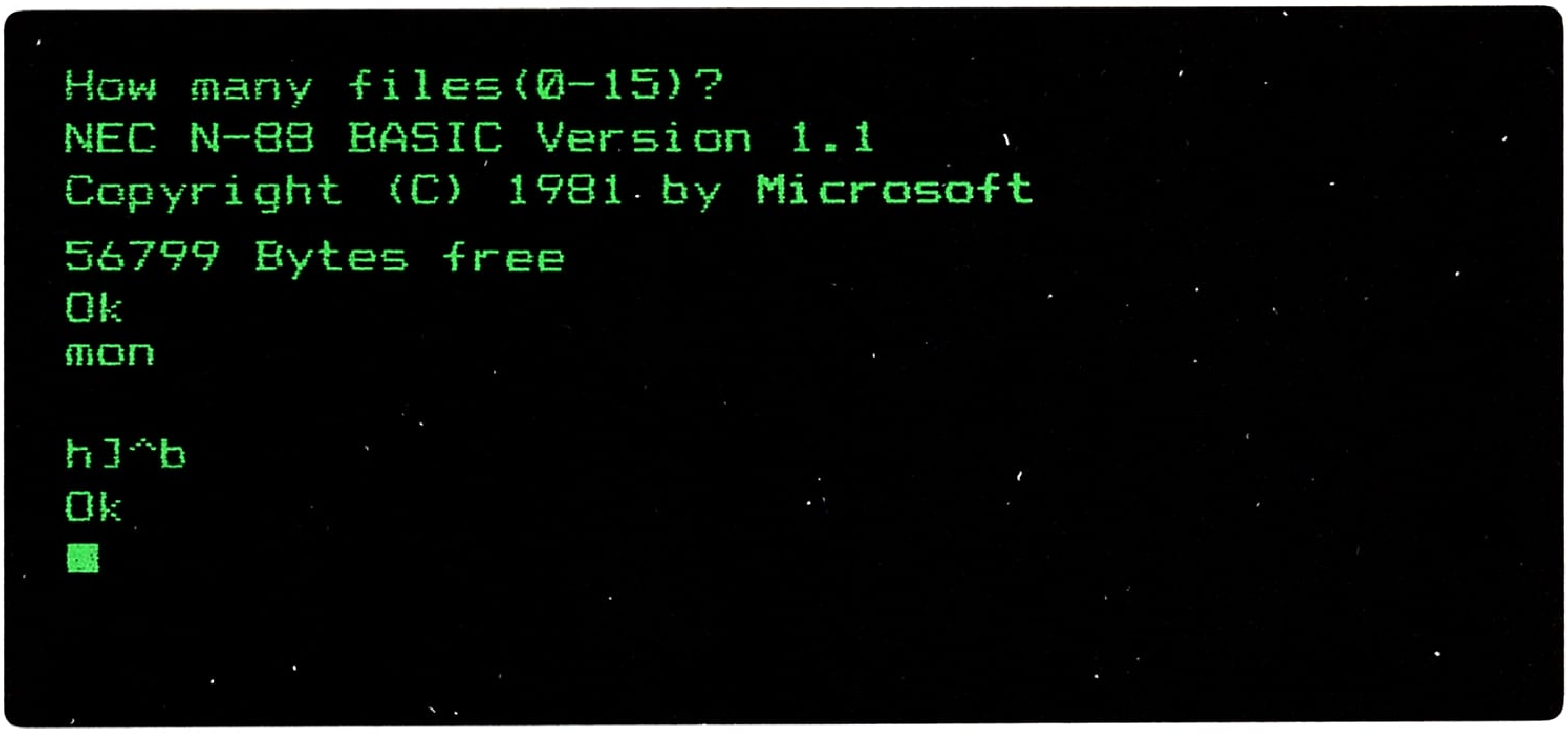
それでは、いよいよマシン語の世界へと踏み込んで行くわけですが、やはり、ＢＡＳＩＣモードへの帰り方を覚えておかなくてはいけないでしょう。

ＢＡＳＩＣへの戻り方は、３通りほどあります。
普通は、

CTRL + B (注)

〈注〉これはコントロールキーを押しながらＢキーを押すことを意味します。

で、戻ります。



しかし、プログラムが**暴走**してしまった時などは、

●「暴走」とは、マシン語プログラムが正常に動作しないときに使う表現です。

STOP キーを押しながらリセットをかける『ホットスタート』

という方法が用いられます。
これで、大抵の場合ＢＡＳＩＣへ戻れますが、どうしても戻らない最悪の事態には、**『コールドスタート』**といって

- **リセットをかける。**
- **電源を１度切って入れなおす。**

などが用いられます。
ただし、この場合プログラムは壊れてしまいます。

1・3 DでLIST

ＢＡＳＩＣでもマシン語でも基本的にはほとんどそっくりです。

ただＢＡＳＩＣの方が人間的で、マシン語の方が機械的な感じがすることが違うぐらいです。

だから、マシン語のコマンドや命令として出てくるほとんどすべてのものは**ＢＡＳＩＣのそれに置き変えて考えることが可能**ですので、

『これから新しい「マシン語」という言語を学ぼう』

と決心しなくても、

『ＢＡＳＩＣの復習をしよう』

ぐらいの軽い気持ちで十分理解できます。

D（ダンプ・メモリ）

最初は、**Ｄコマンド**です。
これは、ＢＡＳＩＣでいう『ＬＩＳＴ』にあたります。

D（表示開始アドレス），（表示終了アドレス）⏎

ＢＡＳＩＣでは、

```
list ⏎
```

とするだけですべてを表示してくれましたが、マシン語では表示するアドレスを決めてやらなければいけません。

```
]d ⏎
```

とするだけでは、

```
0000 F3 31 A0 E1 C3 F7 77 00  ｮ1  ﾄﾃ ｮw
0008 7E E3 BE 23 E3 C2 93 03   ｺｾ#ｺｯﾄ
```

これだけしか表示してくれないからです。

たとえば、

```
hJdb980,b9af
B980 7E C6 EB 32 B7 B9 3A 0C   ニ■2キケ:
B988 BA FE 00 20 12 3E 03 32  コ     > 2
B990 0C BA 3A 0B BA 3D 20 02   コ: コ=
B998 3E 04 32 0B BA 18 04 3D  > 2 コ   =
B9A0 32 0C BA 06 C8 11 00 C0  2 コ ネ   タ
B9A8 F3 1A 67 13 1A 6F 1B 0E  月 g   o
hJ■
```

このように使うのです。

〈注〉上の画面は40桁モードの場合です。

しかし、

```
hJd0000,ffff
```

などとした場合、表示内容が多すぎるため1画面におさまりきらずにスクロールアップしていきます。

●これからよく使いますので16進数、10進数、2進数の対照表を次に示します。

16進数	10進数	2進数
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

途中で一時停止させないとなにが表示されているのかさえわかりません。

そのような時は、ＢＡＳＩＣのＬＩＳＴの途中で一時停止をかけるのと同じく、

CTRL + **S**

と入力し、**一時停止**させます。

このあと**他のキーを押すことによって再びダンプ・メモリが続けられます。**

ダンプ・メモリを中止したい場合は、これもＢＡＳＩＣの場合とまったく同じで、

CTRL + **C**

もしくは、こうします。

STOP

1·4　Eで入力

今度は、マシン語のプログラムを入力してみましょう。

ＢＡＳＩＣの場合では、行番号を打ってそのあとにステートメントを書き、そして ⏎ として１行１行入力していました。

たとえば次のように……。

```
10 PRINT "tomorrow" ⏎
```

しかし、マシン語の場合はこのようにはいかないのです。

まず、『行番号』というものがありません。

アドレス

かわりに、**アドレス**というものがあり、ＢＡＳＩＣで言うところの行番号のような働きをしています。

プログラムを実行させた時、このアドレスの小さい方に書かれた命令から順に行ってゆくのです。

また、このアドレスを目標にプログラム中でＬＯＯＰやＧＯＴＯが行なわれたりもします。

ただ、アドレスは、**００００からＦＦＦＦまでの４ケタの16進数**ですでに用意されており、私たちはこのすでに順番に並んでいるアドレスにプログラムをあとから書き込んでゆくのです。

ここにさきほどのＤコマンドを使ってマシン語のプログラムを出してみました。

```
B000 AF 21 C8 F3 06 19 C5 E5   ｯ!ﾈ月  ﾅ◣
B008 06 50 77 23 3C 10 FB 01    Pw#<  町
B010 78 00 E1 09 C1 10 EF FF   x  ﾄ ﾁ ＼
```

アドレス　　プログラム　　プログラムをアスキーコードになおしたもの

一番左の４ケタの16進数がアドレスです。

まん中の２ケタずつの16進数がプログラムで、一番右の記号は、このプログラムをアスキーコードになおしたらどうなるかを表示したものです。

〈注〉16進数ですので、ＢＯＯ８＋８＝ＢＯ１Ｏとなります。

これを見るとアドレスは、８とびに並んでいますが**(注)**、実はこれは、８列にならんだ２ケタのプログラムの一番左のアドレスを表示し

ているだけなのです。
　本当は、この２ケタの16進数のプログラム１つ１つにアドレスがそれぞれついているのです。

B000	AF	21	C8	F3	06	19	C5	E5
	(B000)	(B001)	(B002)	(B003)	(B004)	(B005)	(B006)	(B007)
B008	06	50	77	23	3C	10	FB	01
	(B008)	(B009)	(B00A)	(B00B)	(B00C)	(B00D)	(B00E)	(B00F)
B010	78	00	E1	09	C1	10	EF	FF
	(B010)	(B011)	(B012)	(B013)	(B014)	(B015)	(B016)	(B017)

E　（エディト・メモリ）

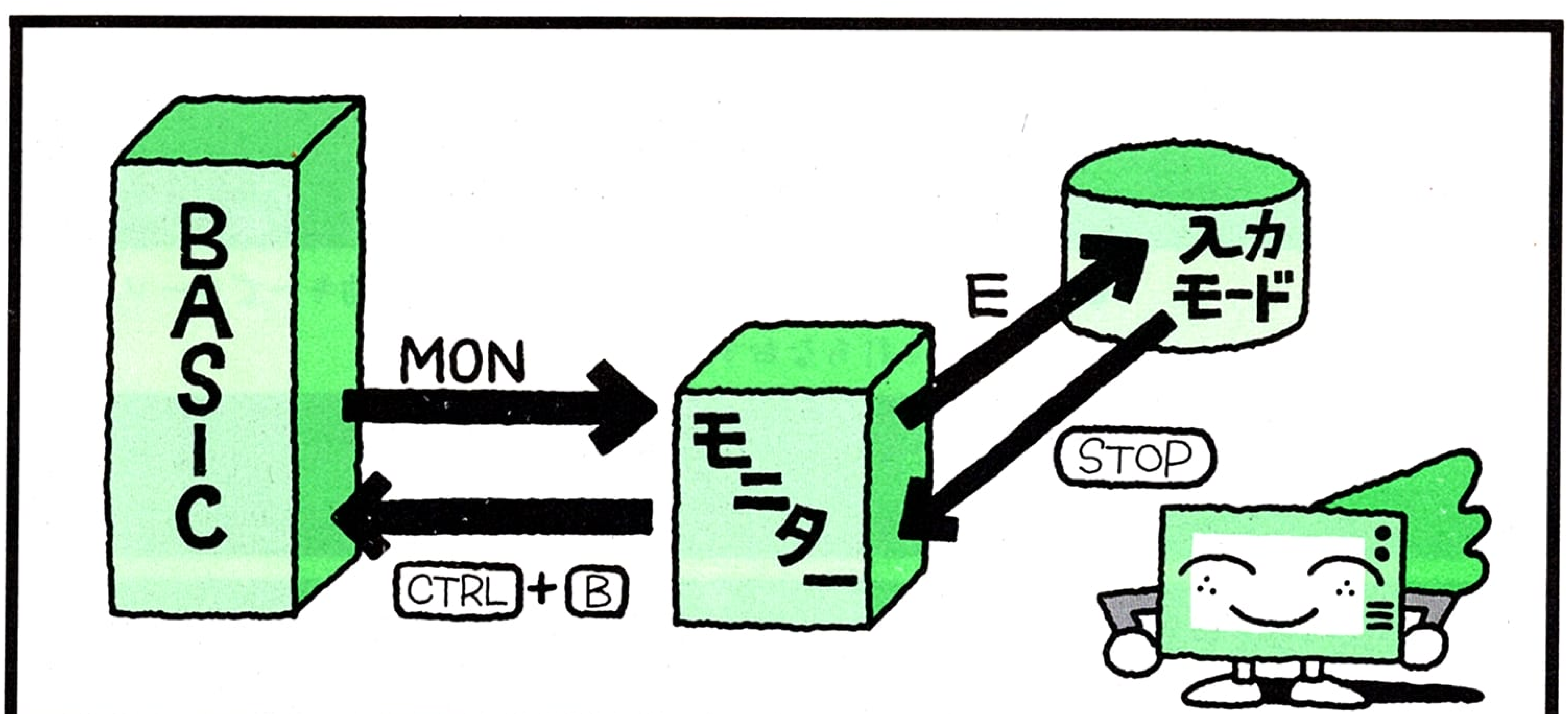
　それではさっそくこのプログラムを入力してみましょう。
　プログラムを入力するためには『入力するモード』にしなくてはいけません。
　その切りかえる命令が、

E　〔開始アドレス〕

です。
　今は、Ｂ０００から入力しますので、

```
h]eb000 ⏎
```

とします。



```
B000 █0 FF 00 FF 00 FF 00 FF
B008 00 FF 00 FF 00 FF 00 FF
B010 00 FF 00 FF 00 FF 00 FF
B018 00 FF 00 FF 00 FF 00 FF
B020 00 FF 00 FF 00 FF 00 FF
B028 00 FF 00 FF 00 FF 00 FF
B030 00 FF 00 FF 00 FF 00 FF
B038 00 FF 00 FF 00 FF 00 FF
B040 00 FF 00 FF 00 FF 00 FF
B048 00 FF 00 FF 00 FF 00 FF
B050 00 FF 00 FF 00 FF 00 FF
B058 00 FF 00 FF 00 FF 00 FF
B060 00 FF 00 FF 00 FF 00 FF
B068 00 FF 00 FF 00 FF 00 FF
B070 00 FF 00 FF 00 FF 00 FF
B078 00 FF 00 FF 00 FF 00 FF
```

〈注〉上の画面は40字モードの場合です。

画面はこのようになり、カーソルはＢ０００の横で点滅しています。

これで入力モードになりました。
あとは、

A F 2 1 C 8 F 3 ……

と、プログラムをそのまま打つだけで、勝手に入力されて行きます。
リターンキーを押す必要もありません。

```
B000 AF 21 C8 F3 █0 FF 00 FF  ｯ!ﾈ月
```

もし打ち間違えた時は、そこまでカーソル移動キーでカーソルを移動させ、正しく打ちなおすだけで修正できます。

```
B000 AF 21 C8 F4 █0 FF 00 FF  ｯ!ﾈ月
```

```
B000 AF 21 C8 F█ 00 FF 00 FF  ｯ!ﾈ月
```

今回は、たった3行だけですのでそんなことはありませんが、普通たくさんのプログラムを打ち込む場合は、画面上に表示されているアドレスだけではたりなくなってしまいます。

そこで、キーボードの上の方についている

ROLL UP　　ROLL DOWN

を使います。

(ROLLUP) を押すと**スクロールアップ**して、**下から続きのアドレスが出て来ます。**
そして、そこへプログラムの続きを書き込むわけです。

さっきのアドレスを見たくなれば、(ROLLDOWN) を押して、**スクロールダウン**させ、さっきのアドレスを見ることができます。

16進数を見た通りに打ち込むだけですから、編集もしやすいですし、入力する方法としては、ＢＡＳＩＣよりやりやすいと言えるでしょう。

プログラムを入力し終えたら、**モニタのモード**へ戻ります。
戻り方は、

STOP

または、

CTRL + C

または、

ESC

です。

1・5　GでRUN

それでは、今入力したばかりの、

```
B000 AF 21 C8 F3 06 19 C5 E5   ｯ!ﾈ月  ｽ◣
B008 06 50 77 23 3C 10 FB 01   .Pw#<  町
B010 78 00 E1 09 C1 10 EF FF   x  ﾄ ﾁ  ＼
```

を実行させてみましょう。

BASICでは、

```
run ⏎
```

ですが、

マシン語では、

G（実行開始アドレス）

とします。

BASICで言うと、ちょうど行番号10から始まるプログラムを

```
goto 10 ⏎
```

として実行させるのと似ています。

●右のページの説明！
なぜ画面をいろんなキャラクタでうめつくしたのでしょうか。
それは、もし１つのキャラクタでうめつくすだけだったらN－BASICモードで、
Color，233⏎
として HOMECLR キーを押してもできるからなのです。

マシン語は、電源を入れた時からすでに００００からＦＦＦＦまでぎっしり何かが書かれています。

それは、プログラムであるかも知れませんし、そうでないかもわかりません。

私たちは、その上から自分の好きなプログラムを書き込むわけですから、普通で言うと、プログラムを書き換えていると言うことになります。

しかし、中には絶対書き換えてはいけないプログラムをあります。(が、そんな所は**ROM**になっていますので、実際私たちが書き込んだりできないようにしてあります。)

私たちが使っていいのは、だいたい**８８３０～Ｅ５ＦＦ**ぐらいの間です。

ただし、この領域は同時にＢＡＳＩＣのプログラムを残しておくためにも使われます。

したがって、私たちがＢＡＳＩＣも使うことも考えて、**Ｂ０００～Ｅ５ＦＦ**ぐらいがいいでしょう。

プログラムはアドレスの最初（００００）から入れることができず途中からしか入力できません。だから、**実行開始アドレスを指定**してやらなくてはいけないのです。

このプログラムはＢ０００から始めますので、

```
hJgb000
```

とします。

```
console 0,25,0,1
width 80,25
```

と、ＢＡＳＩＣで入力してから実行した方がマシン語の威力がよくわかりますね。



1・6 WでSAVE

BASICでは、

```
save "ファイルメイ" ⏎
```

とするだけで、プログラムをSAVEすることができました。

しかし、マシン語はもう少し手を加えなければなりません。

どこからどこまでのアドレスをセーブするのか指定してやらないといけないわけです。

〈注〉本節でのセーブとロードはカセットテープ使用の場合です。ディスクベースでの方法は第6章末で説明します。

W 〔ファイル名〕,〔開始アドレス〕,〔終了アドレス〕

今回は、B000からB017ですので、

```
h]wファイルメイ,b000,b017 ⏎
```

こうします。

もちろん、この前に**カセットの録音ボタンを押しておく**のは、言うまでもありません。

V（ベリファイ・テープ）

〈注〉ベリファイとは、正しくセーブできたか確認することをいいます。

VERIFYするときは、

V 〔ファイル名〕

です。

```
h]vファイルメイ ⏎
```

しかし、

```
h]v ⏎
```

としただけでも、VERIFYは行えます。

ただし、この時は最初に見つけたファイルとの比較を行いますので**テープの頭出しをちゃんとしておきます。**

R（リード・テープ）

では、今度は、今セーブしたプログラムをロードしてみましょう。

ＢＡＳＩＣでは、

```
load "ファイルメイ" ↵
```

だけですみました。

マシン語でも、

R〔ファイル名〕

だけですから、

```
h]rファイルメイ ↵
```

でいいのです。

見つけると

```
Found:ファイルメイ ↵
```

と出て、その下でキャラクタがまるでスロットマシンのようにめまぐるしく表示され、しばらくしたところでカチッという音とともにロードが完了します。

また、プログラムの頭出しさえきちんとできていれば、

```
h]r ↵
```

とするだけでもロードされます。

ただし、この時、『Ｆｏｕｎｄ』は出ません。

1・7 HELP で確認

いちおう、これだけのコマンドを知っていればもうマシン語を使うには十分でしょう。

●**D（ダンプ・メモリ）**

d〔開始アドレス〕,〔終了アドレス〕

●**E（エディト・メモリ）**

e〔開始アドレス〕

●**G（ゴー）**

g〔開始アドレス〕

●**W（ライト・テープ）**

w〔ファイル名〕,〔開始アドレス〕,〔終了アドレス〕

●**V（ベリファイ・テープ）**

v〔ファイル名〕

●**R（リード・テープ）**

r〔ファイル名〕

たったこれだけですのですぐに覚えてしまえると思います。

しかし、うっかりして忘れてしまうこともあると思います。

そんな時、いちいち本を出して見なおしてみることも大切なことですが、もっと手っ取り早く調べることもできるのです。

マシン語のモードになっているなら、テンキーの上についている、

HELP

という「**ヘルプキー**」を押します。

すると、画面にいろいろなコマンドの説明が表示されるのです。

または、

CTRL ＋ **A**

でも同じことが行えます。やってみてください。

右ページの上の画面のようになりましたか？

```
Monitor has following commands.
a <address>                    assemble source text lines.
bh or bq                       select radix (hexa decimal or octal).
d <start>,<end>                dump contents of memory.
e <start>                      change contents of memory by screen editor.
f <start>,<end>,<const>        fill memory by the constant.
g <start>,<break1>,<break2>    execute user program.
i <port>                       read input port.
l <start>,<end>                dis-assemble program in memory.
m <start:s>,<end:s>,<start:d>  move contents of memory block(s=src,d=dest).
o <port>,<new value>           output new value to the port.
p                              toggle print switch.
r <file name>                  read cassette file.
s <address>                    substitute memory contents.
tm                             test memory.
v <file name>                  verify cassette file.
w <file name>,<start>,<end>    write cassette file.
x or x <register name>         dump all CPU register or change the register.
CNTL-b                         return to BASIC
h]■
```

そして、 HELP キーはマシン語モードだけではなく、マシン語入力の『**エディトモード**』でも威力を発揮します。

Eコマンドの使用中に使うのです。

```
Edit command has following subcommands.

up arrow      move cursor up.
down arrow    move cursor down.
left arrow    move cursor left.
right arrow   move cursor right.
CTRL-f        move cursor next byte.
CTRL-b        move cursor prev byte.
ROLL UP       roll screen up.
ROLL DOWN     roll screen down.
STOP,ESC      exit from edit.
numbers       change byte on cursor.

Type any key to return to edit...
```

第2章

マシン語の命令

2・1 アドレス

マシン語のプログラムはすべてコンピュータのメモリの中に記憶されます。

マシン語だけではありません。
ＢＡＳＩＣのプログラムも、そして、そのＢＡＳＩＣを動かしているN88－ＢＡＳＩＣ自身さえもメモリの中に記憶されているのです。

しかし、こういろいろなものを記憶している**メモリのしくみ**は、いったいどうなっているのでしょう。

普通に考えると、それぞれが別々のメモリに記憶されていると思うでしょう。
しかし、メモリは１つしかありません。
すると、１つのメモリの中で範囲を決めて、それぞれのプログラムを分けるようにして記憶していると考えられます。

実は、そうなっているのです。

メモリの中を、「**ここからここまではN88－ＢＡＳＩＣ**」というように範囲を決めて記憶しているのです。
しかし、この場合「ここからここまで」というからにはその場所を明確に表わさなければなりません。
また、そのプログラムの大きさを見るために、**基準を決めておく**必要もあります。

それらを解決するのが、

アドレス

なのです。

メモリ全体を、『**バイト**』**(注)** という単位で分け、それぞれに１つずつ順番に番号をつけます。

〈注〉17ページをごらんください。たとえば「Ｂ００２」というアドレス（番地）には「Ｃ８」というデータが入っています。この「Ｃ８」を「１バイトデータ」といいます。15ページの欄外の表を使って、これを２進数になおしてみますと、

```
1100  1000
 C     8
```

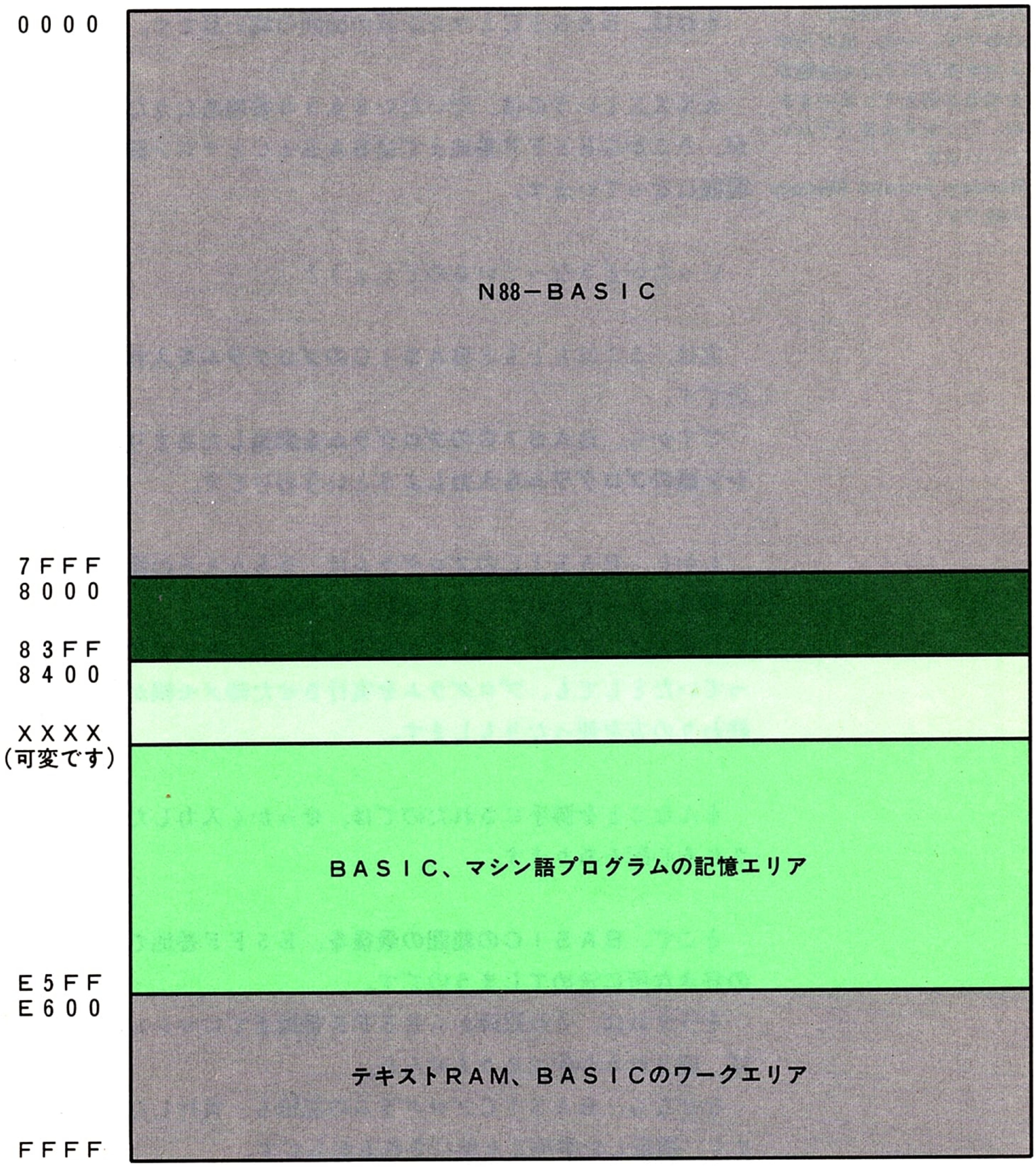
となります。そして、この「１」とか「０」を「１ビット」といい、これがコンピュータの最小単位になります。つまりコンピュータ内部では、このビット単位でいろいろな処理が行われています。しかし私たちが問題にするデータは１バイトデータ、２バイトデータというように、バイト単位で行ないます。いい忘れましたが、上の例からもわかるように、「１バイト」は「８ビット」になります。

こうすれば、「どこに」も明確に、また、プログラムの大きさもはっきり知ることができます。

この番号というのが、実は『**アドレス**』だったわけなのです。

この番号は16進数で、しかも４ケタです。この１つ１つを『アドレス』または『番地』と呼び、ＰＣ－8801シリーズでは『**０００0番地**』から『**ＦＦＦＦ番地**』まであり、その様子は次のようになっています。

●メモリの使用状況を下の図のように示したものをメモリマップといいます。



CLEAR

●電源を切ってもプログラムが消えずに残っている部分をＲＯＭ（ロム）といいます。これは、
Read Only Memory
の略です。一方、私たちが入力するプログラムは電源を切ると消えてしまいますね。ここをＲＡＭ（ラム）といいます。
Random Access Memory
の略です。

これで、なぜマシン語のプログラムを００００番地から入れることができないかおわかりいただけたと思います。

つまり、そこにはすでにN88－ＢＡＳＩＣのプログラムが入っていたからなのですね。（大切なので**ROM**になっています。）

しかし、まだ疑問は少し残ります。

それは、ＢＡＳＩＣとマシン語の範囲の境い目です。

ＸＸＸＸというのは、だいたい８８３０番地あたりだと思うのですが、そこからＥ５ＦＦ番地まではＢＡＳＩＣとマシン語がいっしょの範囲になっています。

いったいどうなっているのでしょう？

実は、ここはもともと**ＢＡＳＩＣのプログラムを入れておく場所**なのです。

ですから、**ＢＡＳＩＣのプログラムを記憶したあまりのメモリにマシン語のプログラムを入力しよう**というわけです。

しかし、ＢＡＳＩＣのプログラムは、きちんとその範囲のはじめから順番に入ってくれているとはかぎりません。

たとえば、きちんとはじめからまん中のあたりまでプログラムが入っていたとしても、プログラムを実行させた時メモ帳がわりに範囲の終わりの方を使ったりもします。

そんなことを勝手にされたのでは、せっかく入力したマシン語が壊される心配もあります。

そこで、**ＢＡＳＩＣの範囲の最後を、**Ｅ５ＦＦ番地ではなく、**自分の好きな所に決めてしまう**のです。

そうすれば、それ以降からＥ５ＦＦ番地までにマシン語を書き込めば、壊される心配はありませんね。

なぜなら、ＢＡＳＩＣプログラムの記憶も、実行した時に使うメモリも、指定した番地より前にされるからです。

その命令が、

CLEAR ,〔メモリの上限〕

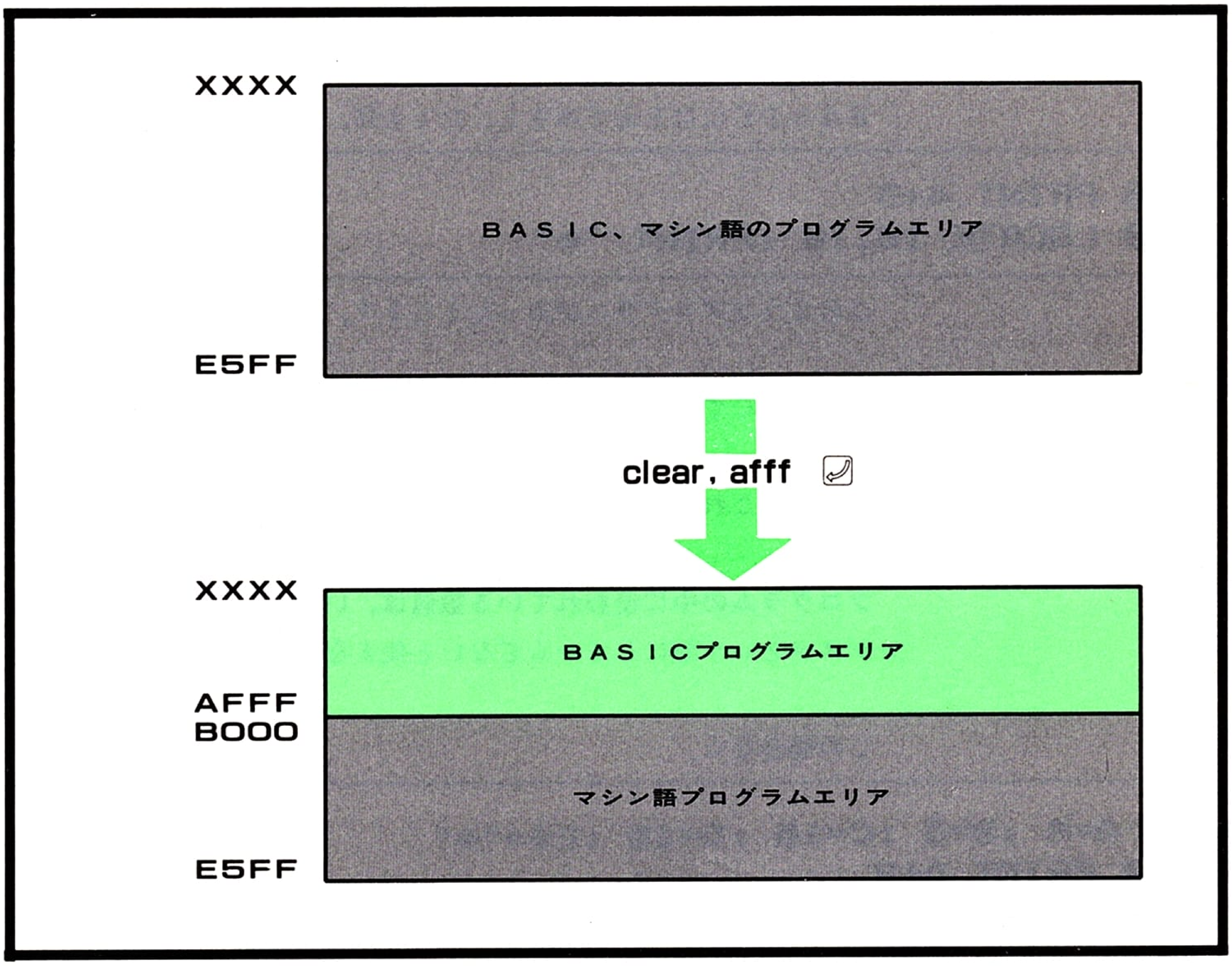
です。

これは、ＢＡＳＩＣのステートメントですので、ＢＡＳＩＣモードにして使います。

たとえば、Ｂ０００番地以降をマシン語のプログラムの入力場所としたいなら、その１つ手前のＡＦＦＦ番地までをＢＡＳＩＣプログラムの範囲とするわけです。

```
clear,&hafff ⏎
```

と打ち込みます。



もし、オールマシン語にするなら必要ありませんが、いつＢＡＳＩＣを使うとも限りません。

ＣＬＥＡＲする癖をつけておいた方がいいでしょう。

2・2 レジスタ

ＢＡＳＩＣで言うところの**『変数』と同じ働きをするもの**が、マシン語にもあります。

レジスタ

です。

しかし、このレジスタは、ＢＡＳＩＣの変数よりもずっと重要なものです。
演算やほとんどすべての処理は、このレジスタがないと働かないのです。

ＢＡＳＩＣにたとえてみると、たとえば、

```
10 PRINT 4+3
20 LOCATE 10,10 :PRINT "♥"
```

このようなプログラムがあったとします。

ＢＡＳＩＣでは、ごく普通のプログラムですね。

しかし、これがもしマシン語だったとしたら、こんなことはできません。
プログラムの中に使われている数値は、いちどレジスタに代入してレジスタという形にしてからでないと使えないのです。

この場合なら、

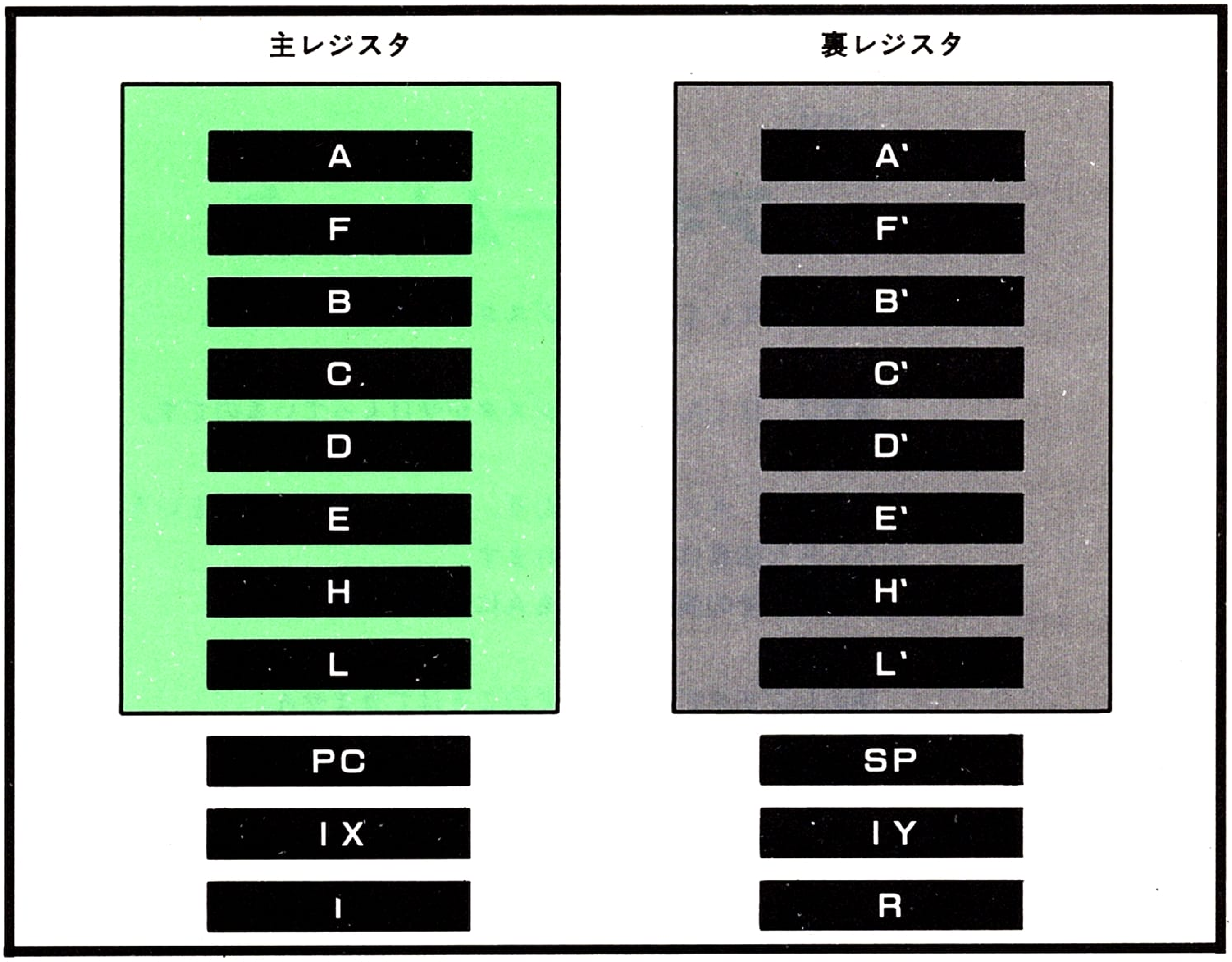
```
5 A=4 :B=3 :C=10 :D=10 :E$="♥"
10 PRINT A+B
20 LOCATE  C, D :PRINT E$
```

このようにしなければならない、というわけです。

レジスタは、マシン語の中で最も重要だと言っていいでしょう。

レジスタの種類

しかし、この大切なレジスタもそれほどたくさんはありません。



上の図に表わしただけしかないのです。

しかも、PC や SP、IX、IY、I、R などはマシン語を実行している時に、CPUもこのレジスタを使用していますので、そう軽々とは使えません。

初心者のうちは使わない方がいいでしょう。

そして、**主レジスタと裏レジスタを同時に使用することもできません。**

命令が使えるのは主レジスタだけで、裏レジスタは主レジスタと、交換しないと使えないのです。

つまり、私たちが普通に使えるのは

A F B C D E H L

の8つだけです。

アキュームレータ

私たちがすぐに使える８つのレジスタの中で、最も大切なレジスタが**Ａレジスタ**です。

これは、

アキュームレータ

とも呼ばれていて、他のレジスタと少々違っています。

演算は、ほとんどこのレジスタが受けもっているのです。

たとえば、ＡにＢを加えたり、ＡからＨを引いたりというように、いろいろな演算はＡと行われます。
そして、**その演算の結果もＡに代入されます。**

他のレジスタではこれほどのことはできません。

ですから、プログラムを作る時、Ａレジスタはいつでも自由に使えるように、数値の保存などには使わない方がいいでしょう。

フラグ・レジスタ

アキュームレータと同じぐらい大切なレジスタに、**Ｆレジスタ**があります。

これは、

フラグ・レジスタ

と呼ばれています。

フラグ・レジスタは、私たちが勝手に数値を代入したりできません。
演算などを行った時に、**ＣＰＵがその結果に応じて、中身を書き換えてしまう**のです。

私たちは、ＣＰＵが書き換えたフラグ・レジスタを見て、

「今の足し算はくり上がりがあった」
「引き算をしたら、０になった」

などと、演算の状況を知ることができるのです。

汎用レジスタ

残りの６つのレジスタは、そのままの形でレジスタとして使うことが当然できます。

ただし、それぞれには１バイトの数しか入れることができません。
００～ＦＦまでしか入れられないというわけです。

10進数になおすと、**0～255 まで**、ということになり、たったこれだけでは何の計算もできないのと同じです。

●１番大きな１バイトデータは「ＦＦ」です。これは16進数表記ですから、10進数になおすには
　Ｆ×16＋Ｆ
を計算すればよいのですが、16進数の「Ｆ」は10進数で「15」ですから、これをＦのところに代入して
　15×16＋15＝255
となります。

そこで、ＢとＣ，ＤとＥ，ＨとＬをつなげて１つのレジスタとして使うのです。
そうすれば、２バイトですので、００００～ＦＦＦＦ、10進数になおすと、－32768～＋32767までが扱えるわけです。

この２つのレジスタをペアにしたものを、

汎用レジスタ

といいます。

A	**アキュームレータ**（演算やいろいろな処理で最も活躍）
F	（演算の状況を知る）

B	C	
D	E	**汎用レジスタ**（データの一時保存やアドレスの指定など。中でもＨＬレジスタは重要）
H	L	

2・3　Xでレジスタを見る

さて、ＢＡＳＩＣでは「変数」の中身を見たい時は、

```
print a
```

とすれば、すぐに見ることができました。

ＰＲＩＮＴ文によって見ることができるのが、あたり前だったわけです。

ところが、マシン語ではこうはいきません。

マシン語にＰＲＩＮＴ文に相当する命令はないのです。

したがってＰＲＩＮＴしたい時は、ＰＲＩＮＴさせるためのプログラムをいちいち作らなくてはならないのです。

最初からそんなプログラムを作るのはむつかしいことですし、第一レジスタの中身を見るためだけにそんなプログラムを入力しなくてはならないとは、非能率的すぎます。

しかしながら、レジスタを見たいだけなら、ＰＣ－8801にはすばらしいコマンドがあります。

X

です。**イグザミン・レジスタ・コマンド**です。

モニタモードで

```
h]x
```

としてみます。

```
A :00 F :PZ---E-- B :0000 D :EDCC
H :0001 A':00 F':P----O-- B':0004
D':FF7B H':0911 IX:0000 IY:8000
I :F3 PC:0000 SP:E5F9
```

このように表示されるのです。

A：00

アキュームレータのことです。中身は００だとわかります。

F：PZ---E--

フラグ・レジスタです。**この記号が演算の状況を示しています。**

B：0000

ＢＣレジスタです。この場合はＢレジスタもＣレジスタも００です。

D：EDCC

ＤＥレジスタです。最初の２ケタがＤレジスタ、終わりの２ケタがＥレジスタを表わしています。

この場合ＤレジスタはＥＤ、ＥレジスタはＣＣです。

H：0001

ＨＬレジスタです。ＨＬレジスタは００、Ｌレジスタは０１です。

A', F', B', D', H'

これらは**裏レジスタ**です。

IX，IY

これらは**インデックス・レジスタ**です。

I

インタラプト・ベクタレジスタというレジスタです。

PC

プログラム・カウンタです。簡単に言うと、現在実行しているアドレスの次のアドレスを示しているレジスタです。

SP

スタック・ポインタというレジスタです。

このＸコマンドは、これからプログラムを作ったりしていく上で、非常に便利なコマンドとなります。

しっかり覚えておきましょう。

2・4　ハンド・アセンブル

マシン語の命令はすべて２ケタの16進数でできています。

```
B940 F3 ED 73 70 B9 31 74 B9  月□spケ1tケ
B948 D3 5C CD 5F B9 D3 5D CD  モ¥ヘ_ケモ]ヘ
B950 5F B9 D3 5E CD 5F B9 D3  _ケモ^ヘ_ケモ
B958 5F ED 7B 70 B9 FB C9 AF  _□{pケ町ノッ
B960 32 00 C0 01 7F 3E 11 01  2 タ>
B968 C0 21 00 C0 ED B0 C9 00  タ! タローノ
```

これがマシン語のプログラムですね。

しかし、こういうのがマシン語のプログラムだ、ということを知っているからわかったことであって、何も知らないでこれを見たとしたらどうでしょうか。

きっと、コンピュータのデータか何かの暗号だと思うはずです。

そうです。その第六感は鋭い所をついています。

マシン語は、確かに２ケタの16進数だったわけですが、実はこれは、ＣＰＵが理解しやすい表現方法で表わしているからであって、それぞれの数値にはちゃんとした意味が含まれているのです。

つまり、**ＣＰＵ用の暗号**、というわけです。

暗号ですから、当然私たちにもわかるように訳すことができるはずです。

このことを一般に、

逆アセンブル

と言い、その私たちにもわかるような形にしたものを、

ニーモニック

と言います。

たとえば、

```
B000   3E 04 06 03 80
```

というプログラムは、**逆アセンブルして、ニーモニックに訳すと、**

```
B000   LD    A, 4
B002   LD    B, 3
B004   ADD   A, B
```

となるのです。

ここまでくればもうＢＡＳＩＣもマシン語もたいした違いはありません。

一見しただけで何をしたいプログラムなのかだいたいわかります。

ちなみに、これをＢＡＳＩＣに訳してみると、

```
10 A=4
20 B=3
30 A=A+B
```

となり、ニーモニックのそれとほとんどそっくりです。

暗号の解読さえできれば、もうマシン語もＢＡＳＩＣ感覚で理解できるのです。

逆アセンブル

それでは、暗号の解読方法を御説明いたしましょう。

といって、そんなに大げさなことではありません。
答は簡単。
「虎の巻」を見るのです。

マシン語では、**「機械語→ニーモニック対応表」「ニーモニック→機械語対照表」**がそれにあたります。

この本のおしまいに付録としてのせてあります。

まず最初が『**3 E**』ですから、「機械語→ニーモニック対応表」で３Ｅの所をさがします。
するとそこには、

3E	**LD**	**A, n**

とありました。

「**3 E**」**は**「**L D　A, n**」**の意味**だということです。

しかし、ここで注意しなくてはいけないことがあります。
それは、「ｎ」となっている所です。

「ｎ」というのは私たちがプログラムを作る時に指定する数値で、その命令の後ろにつけるものなのです。

ですから、この場合だと「**3 E　0 4**」となっていますので、**この**「**0 4**」**がｎにあたる**というわけです。
つまり、「３Ｅ　０４」の２バイトがあって始めて、

LD　A, 4

と訳せるわけなのです。
「３Ｅ　０４」で１つの文ですから、**次の命令は**「**0 6**」ということになります。
「０６」も同じように「ＬＤ　Ｂ，ｎ」という意味なので、その次の０３がｎの値ということになり、「**0 6　0 3**」**で**「**L D　B, 3**」**と訳すことができます。**

次は「８０」で、これは「ＡＤＤ　Ａ，Ｂ」となっているので、そのまま訳して「**A D D　A, B**」です。

これが逆アセンブルなのです。ただ機械的に訳しているだけなので、最初はおもしろくても、いつかはあきてきます。
だから普通は「**逆アセンブラ**」というプログラムに訳をさせたりします。

〈注〉「逆アセンブラ」とは「3E　04　06　03　80」といったマシン語プログラムを「LD」と「ADD」などのニーモニックを使った形になおしてくれるプログラムのことです。

でも最初のうちは自分の力で解いてみる方が楽しいのですから、自力で解く方がいいと思います。勉強にもなることですし……。

ハンド・アセンブル

しかし、人が作ったプログラムを訳しているだけではなんにもなりません。実際に自分で作れるようになりたいものですね。

作る場合は、いま私たちが機械語プログラムをニーモニックに訳しましたが、今度はこれを逆にたどればよいのです。つまり、私たちがとても理解しやすい**ニーモニックでプログラムを作る**のです。

そして、そのあとで「**ニーモニック→機械語対照表」を使って、16進数のマシン語に訳す**のです。

この作業を

アセンブル

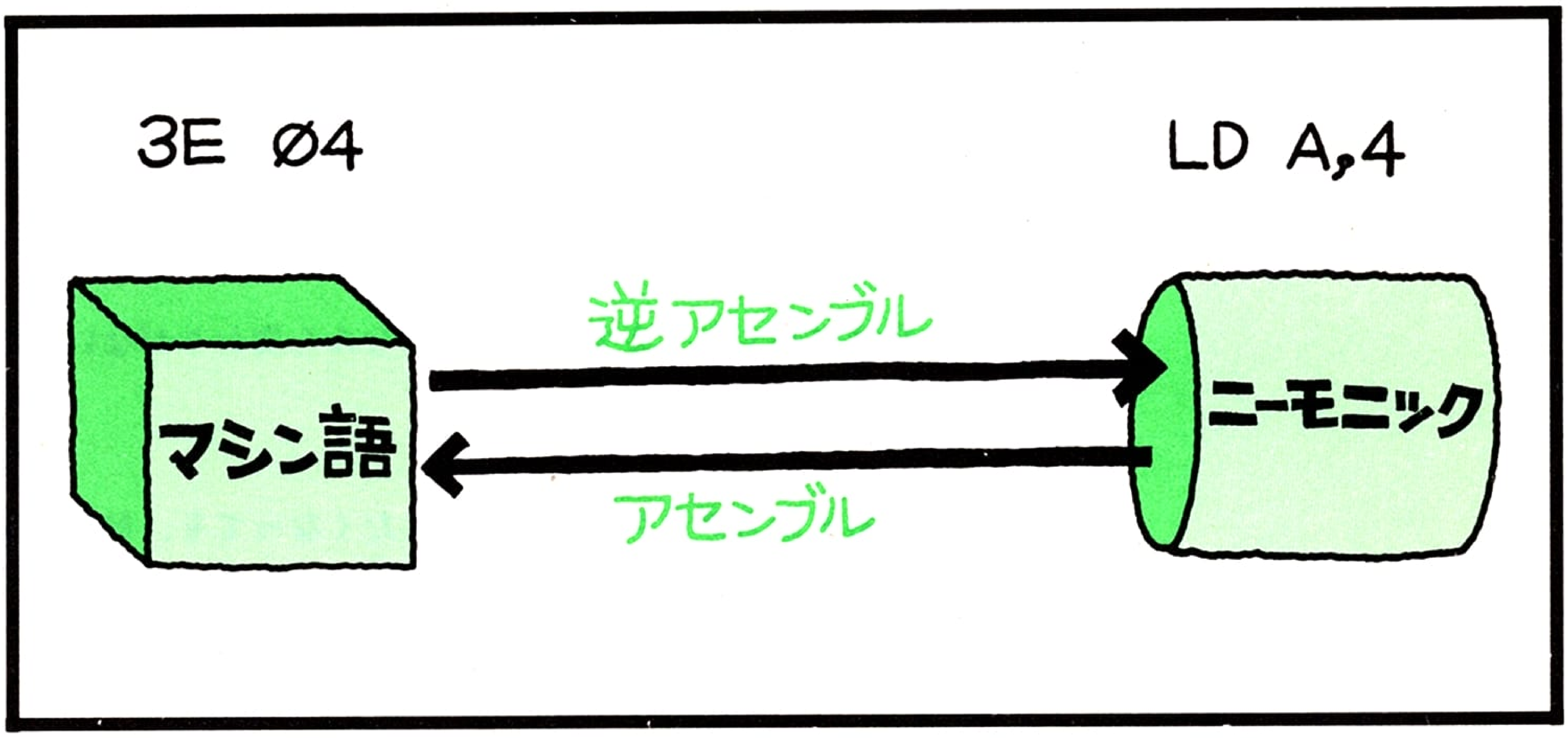
と言い、私たちが表を見ながら訳すことを、

ハンド・アセンブル

と言います。

そして、この作業は手間がかかりますので「アセンブラ」という、人間のかわりに訳してくれる「かしこいプログラム」を使うにこしたことはありませんが、なくても別にかまいません。

というより、初心者のうちはハンド・アセンブルする方が命令をいろいろと覚えられるのでいいでしょう。



2·5 NOP 命令

それでは、いよいよこれからマシン語の命令を説明していくことにいたしましょう。

最初は、

NOP
（ノンオペレーション）

NOP

です。

これは、**「何もしない」**命令です。

何もしないのに、なぜこんな命令があるのでしょう。

実は、**何もしないからこそ大変便利だ**ということもあるのです。

たとえば、ＢＡＳＩＣでプログラムを作った時、普通行番号は10番きざみにつけてあります。
それは、プログラムを追加したくなった時、あとから間に入れることができるようにそうしていたのです。

しかしマシン語の場合、**アドレスは１つずつ順番につけられています。**
ですから、あとからちょっとプログラムを追加するようなことができません。

しかし、**プログラムの途中に何もしないＮＯＰ命令を入れておいたらどうなるでしょう。**

プログラム自体は長くなりますが、働きはなんら変わりがないはずです。

こうすれば、あとからプログラムを追加したくなっても、**ＮＯＰ命令を追加したいプログラムに書き変えればよい**というわけです。

そして反対に、いらない命令を取り去る時も、ＢＡＳＩＣのように、デリートキーでちょっと消すなんてことはマシン語にはできません。

そんな時もＮＯＰ命令は活躍します。

いらなくなった命令をＮＯＰ命令に書き変えればいいのです。
そうすれば、その命令を消したのと同じ結果が得られるはずなのですから……。

RST 38H

では、次は

RST 38H

です。

RST　38H
（リスタート）
〈注〉この命令はＰＣ－8801の時のみ有効です。

これは、マシン語モードのＧコマンドでマシン語を実行した時、そのままではいつまでたってもマシン語の実行は終わらないので、**モニタモードに戻してプログラムを止める**という命令なのです。

ただし、今は、やむを得ず使っていますが、本当は、ちゃんとしたプログラムを作る時には使わない方がいいと思います。

ＮＯＰはマシン語では「００」です。
そして、ＲＳＴ　３８Ｈは「ＦＦ」です。

ですから、

アドレス	マシン語	ラベル	ニーモニック	コメント
B000	00		NOP	何もしない
B001	FF		RST 38H	モニタへ戻る

というプログラムがあったとします。

```
h]gb000 ⏎
```

で実行されますが、実際は何も起こりませんね。
ＲＳＴ　３８Ｈは『ＥＮＤ』だと考えてもいいでしょう。

2·6　LD命令(Ⅰ)

これはマシン語の命令の中でも最も重要で、また最もよく使う命令です。

ＢＡＳＩＣで言うと、**「ＬＥＴ文」**にあてはまりますが、時には、**「ＰＥＥＫ」**や**「ＰＯＫＥ」**にもなります。

まずは、

ＬＤ　ｒ，ｒ'
（ロード）

ＬＤ　レジスタ，レジスタ

です。

これは、もし**「ＬＤ　Ａ，Ｂ」**という場合で考えると、

「Ａレジスタに Ｂレジスタの内容を代入する」

ということで、つまり、レジスタからレジスタへの代入文という命令なのです。

ＬＤ　ｒ，ｎ

ＬＤ　レジスタ，数値

今度もほぼ同じ形ですが、レジスタに代入するのはレジスタの「内容」ではなく、私たちが指定する**「数値」**なのです。

たとえば、

「ＬＤ　Ａ，４」

のように使います。

ＢＡＳＩＣで言うなら、『ＬＥＴ文』にあたりますので、

```
LD   A,B     --->    LET  A=B

LD   A,4     --->    LET  A=4
```

といったところでしょうか。

★for example★

それでは例として、A，B，C，D，E，H，Lのレジスタを０に**するプログラム**を作ってみます。

それぞれのレジスタに００を代入していってもいいですし、また、１つのレジスタに００を代入しておいて、そしてそれを他のレジスタに代入していってもできますね。

それでは、後者を使ってみましょう。

アドレス	マシン語	ラベル	ニーモニック	コメント(覚え書き)
			LD A, 00	A←0
			LD B, A	B←A
			LD C, A	C←A
			LD D, A	D←A
			LD E, A	E←A
			LD H, A	H←A
			LD L, A	L←A
			RST 38H	モニタに戻る

というプログラムになることがわかります。

では、今度は、プログラムの先頭番地を決めて**ハンド・アセンブル**します。そのあと、順番にアドレスをつければ、

アドレス	マシン語	ラベル	ニーモニック	コメント
B000	3E 00		LD A, 00	A←0
B002	47		LD B, A	B←A
B003	4F		LD C, A	C←A
B004	57		LD D, A	D←A
B005	5F		LD E, A	E←A
B006	67		LD H, A	H←A
B007	6F		LD L, A	L←A
B008	FF		RST 38H	モニタに戻る

となるのです。

それでは入力です。

入力するのは、マシン語の部分だけでいいのですから、

アドレス	マシン語	ラベル	ニーモニック	コメント
B000	3E 00		LD A, 00	A←0
B002	47		LD B, A	B←A
B003	4F		LD C, A	C←A
B004	57		LD D, A	D←A
B005	5F		LD E, A	E←A
B006	67		LD H, A	H←A
B007	6F		LD L, A	L←A
B008	FF		RST 38H	モニタに戻る

このようになります。

Eコマンドで入力します。

入力できたら、ストップキーを押して、次に**Dコマンド**（ダンプ・メモリ）で確認してみましょう。

```
B000 3E 00 47 4F 57 5F 67 6F  > GOW_go
B008 FF
```

このようになりました。これがプログラムなのです。

それでは、**Gコマンド**で実行してみましょう。

スタートアドレスはＢ０００ですから、

```
h]gb000 ⏎
```

です。

実行してもすぐマシン語のコマンド入力待ちに戻りますので、見た目では何が起こったのかわかりません。

そこで、**X**でレジスタを見てみましょう。０になっているはずです。

```
A :00 F :PZ---E-- B :0000 D :0000
H :0000 A':00 F':PZ---E-- B':0000
D':FF7B H':0911 IX:0000 IY:8000
I :F3 PC:B008 SP:B6FB
```

16ビットロード

さっきまでは、すべて１つのレジスタへの代入文でした。

しかし、ＢＣ，ＤＥ，ＨＬなどのレジスタペアの時には、

ＬＤ　レジスタペア，数値　　　　ＬＤ　ｄｄ，ｎｎ

という命令を利用して、２ついっぺんに代入することもできるのです。

```
LD    BC, 1234
```

とすれば、**ＢＣレジスタに「１２３４」という16進数が代入されます。**
もちろん、

```
LD    B, 12
LD    C, 34
```

と分けて代入しても同じことですが、レジスタペアとして使用する時は分けずに代入した方がプログラムを見やすくさせます。

ただし、注意すべき点が１つあります。
それは、ハンドアセンブルする時です。

```
LD    BC, 1234
```

をハンドアセンブルする時、「ＬＤ　ＢＣ，ｎｎ」は「０１　ｎｎ」というマシン語になおすことができますが、この時「ｎｎ」に入力する数値は、**上２ケタと下２ケタをひっくり返さないといけない**のです。
この場合なら、「１２」と「３４」を入れ換えて、

```
01  34  12
```

としなければならないということです。
これは、ＣＰＵの構造上こうした方がいいのでこうなってきます。
ですから、４ケタの16進数を扱う他の命令の時もこうします。

2・7　LD命令(II)

今度は、

LD　A，（nn）

LD　A，(アドレス)

というＬＤ命令です。

一見すると、Ａレジスタに「アドレス」を代入するように見えますが、そうではありません。

Ａレジスタに、**ここで指定されているアドレスの「メモリの内容」を代入する**という命令なのです。

メモリの中身をＡレジスタに読み出す命令ですから、ＢＡＳＩＣで言うと、ちょうど「**ＰＥＥＫ**」に相当します。

LD　r，（HL）

LD　レジスタ，(HL)

という命令もあります。
これもほぼ同じ働きの命令です。

レジスタに、**ＨＬレジスタペアで指定されたアドレスの「内容」を代入する**という命令なのです。

ＢＡＳＩＣでたとえてみると、

```
LD   A,(B000)      --->      LET  A=PEEK(&hB000)

LD   B,(HL)        --->      LET  B=PEEK(HL)
```

こうなります。

どちらも、カッコで囲まれた中身は、「代入する内容」ではなく、**「代入するべき数値が保存されているメモリのアドレス」**を示していることを忘れないようにして下さい。

★for example★

まず、Eコマンドで、たとえばB010にAAを入力します。

```
B010 AA FF 00 FF 00 FF 00 FF  I
```

これをAレジスタに代入するプログラムを作ってみましょう。
B010番地のメモリの内容をAレジスタに代入するのですから、

アドレス	マシン語	ラベル	ニーモニック	コメント
			LD　A,(B010)	A←(B010)
			RST　38H	モニタに戻る

こうなります。

そして、プログラムの**スタートアドレスを決める**わけですが、これは、プログラムを作る前に決めてしまうこともできます。

そこで使われるのが、

ORG　スタートアドレス

ORG
(オリジン)

という命令です。

命令といっても、実際にはマシン語にこのような命令はありませんし、まして何かを実行したり処理したりするものでもありません。
プログラムを作る時、**プログラムのスタートアドレスを忘れてしまわないように書きとめておく**だけのものです。
普通、プログラムの先頭につけます。これで、スタートアドレスがどこだか一目でわかりますね。

アドレス	マシン語	ラベル	ニーモニック	コメント
			ORG　B000	
			LD　A,(B010)	A←(B010)
			RST　38H	モニタに戻る

では、**アセンブル**します。

しかし、ＯＲＧはアセンブルする必要はありませんね。実際には存在しない命令なのですから……。

アドレス	マシン語	ラベル	ニーモニック	コメント
			ORG B000	
B000	3A 10 B0		LD A,(B010)	A←(B010)
B003	FF		RST 38H	モニタに戻る

このようになりました。

でも、よく注意して下さい。

「**LD A，(B010)**」は「３Ａ Ｂ０ １０」ではありません。

「**3A 10 B0**」です。

『Ｂ０１０』というアドレスは４ケタの16進数です。４ケタの16進数の場合「ＬＤ ＢＣ，１２３４」の時と同じく、**上2ケタと下2ケタを交換する**のです。

では、**Eコマンドで入力**します。

```
B000 3A 10 B0 FF                    : -
```

次に**Gコマンドで実行**して、

```
h]gb000 ↵
```

Xコマンドでレジスタを見ると、

```
A :AA F :PZ---E-- B :0000 D :EDCC
H :0001 A':00 F':PZ---E-- B':0000
D':FF7B H':0911 IX:0000 IY:8000
I :F3 PC:B003 SP:E5F9
```

確かに、Ａレジスタに「ＡＡ」が入っていることがわかります。

今度は先ほどと同じことをするプログラムを、違う方法で作ってみましょう。

先程は、「ＬＤ　Ａ，（ｎｎ）」を使用しましたが、
今回は、「ＬＤ　ｒ，（ＨＬ）」を使用してみるのです。

ＨＬに『Ｂ０１０』**を代入**すれば、あとはほとんど同じです。

アドレス	マシン語	ラベル	ニーモニック	コメント
			ORG B008	
			LD HL, B010	ＨＬ←Ｂ０１０
			LD A, (HL)	Ａ←（ＨＬ）
			RST 38	モニタに戻る

こうなります。
そしてアセンブルをしますが、『Ｂ０１０』をひっくり返すことに注意しておいて下さい。

アドレス	マシン語	ラベル	ニーモニック	コメント
			ORG B008	
B008	21 10 B0		LD HL, B010	ＨＬ←Ｂ０１０
B00B	7E		LD A, (HL)	Ａ←（ＨＬ）
B00C	FF		RST 38H	モニタに戻る

Ｅコマンドで入力、

```
B008 21 10 B0 7E FF                 ! -~
```

Ｇコマンドで実行、

```
h]gb008
```

Ｘコマンドで、

```
A :AA F :PZ---E-- B :0000 D :EDCC
H :B010 A':32 F':P----O-- B':0000
D':FF7B H':0911 IX:0000 IY:8000
I :F3 PC:B00C SP:E5F9
```

確認します。確かになっていますね。
Ｂ０１０番地の値をいろいろ変えて試してみるのも楽しいでしょう。

16ビットロード

レジスタペアにメモリから代入する命令もあります。

ＬＤ ｄｄ，（ｎｎ）

LD レジスタペア，（アドレス）

です。

これも、「ＬＤ　Ａ，（アドレス）」などと同じように**アドレスの指している「メモリの内容」をレジスタペアに代入**するわけです。

しかし、レジスタペアに代入する数値は４ケタの16進数です。
つまり２バイトの数値を代入しなくてはならないのですが、**指しているアドレスは１か所だけ**です。これでは、１バイトの数値しかわかりませんね。

実は、この命令は**指しているアドレスと、その次のアドレスの数値を代入する**という命令なのです。
ですから、例えば、

ＬＤ　ＨＬ，（Ｂ０１０）

という場合なら、**ＨＬレジスタに、Ｂ０１０番地の数値とその次のＢ０１１番地の数値が代入される**ということなのです。

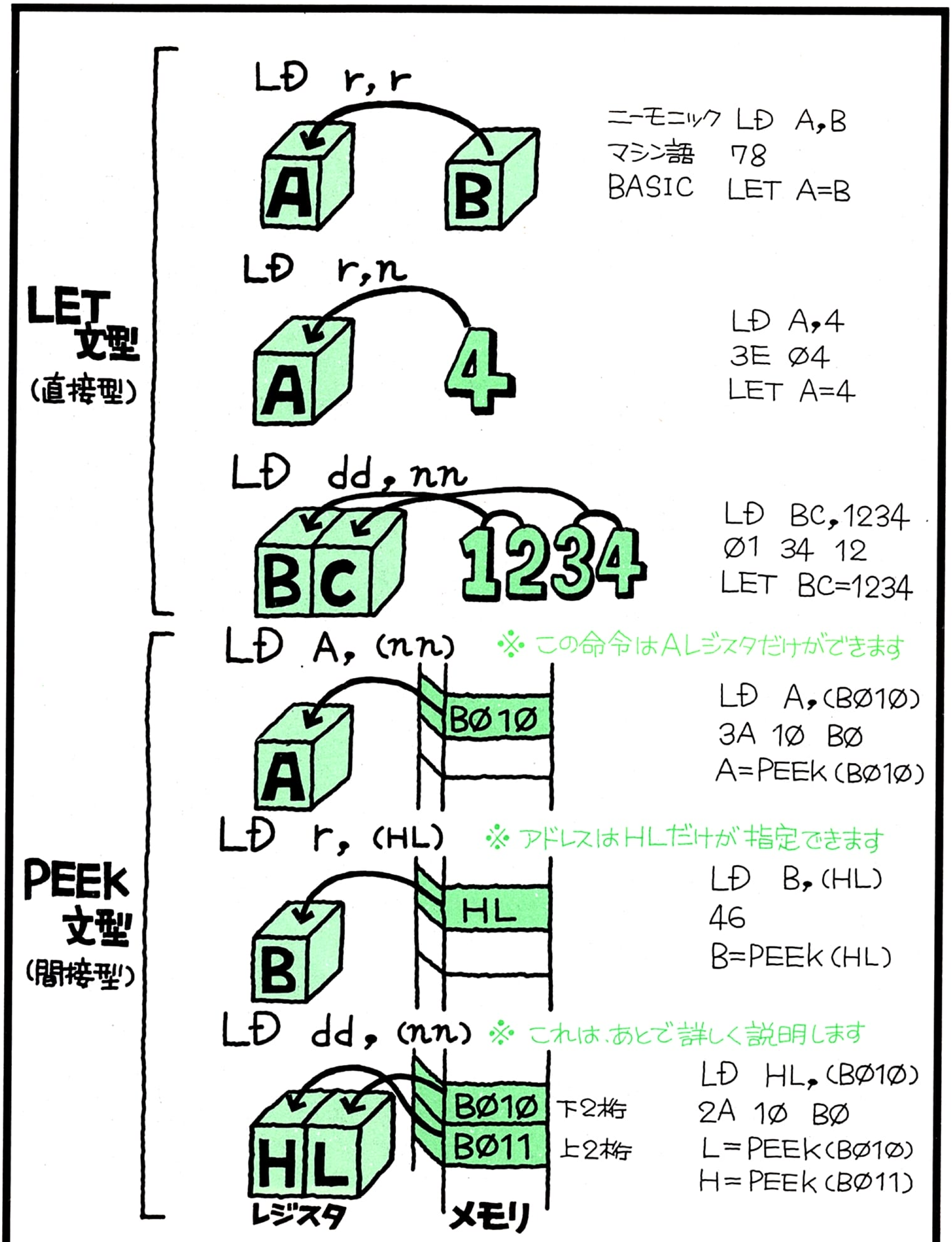
ただし、この場合も４ケタの16進数を扱っていますので、気をつけなければならないことがあります。
もちろん、「ＬＤ　ＨＬ，（Ｂ０１０）」をアセンブルした時に、「２Ａ　１０　Ｂ０」とするのは当然ですが、それだけではありません。

代入する数値をメモリに保存する時もひっくり返す

のです。この場合なら、**ＨにＢ０１１番地の、ＬにＢ０１０番地の数値が代入される**のです。

LDの中間復習

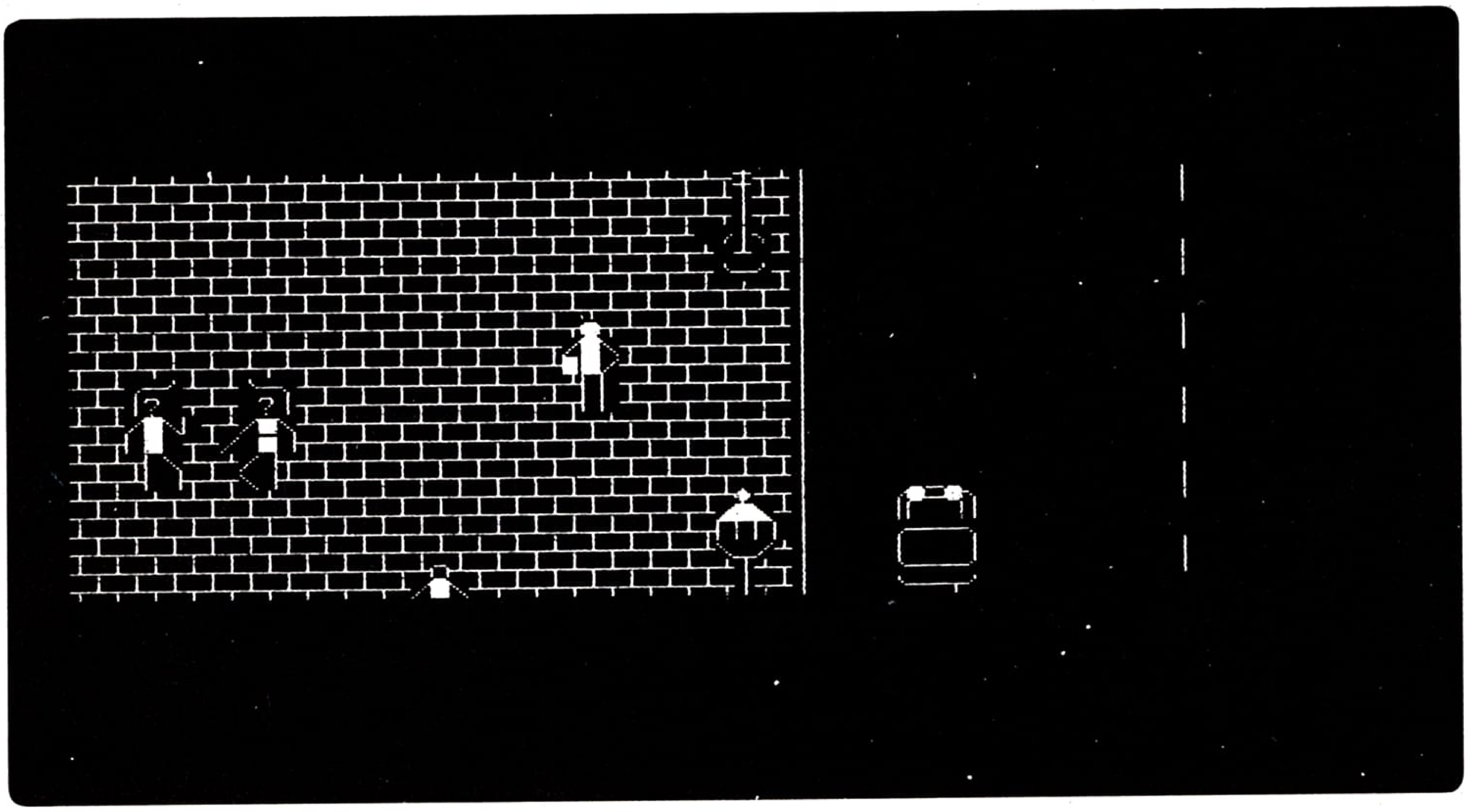
なんだか、いろいろよく似たのがたくさん出てきたので、少しまとめておきます。

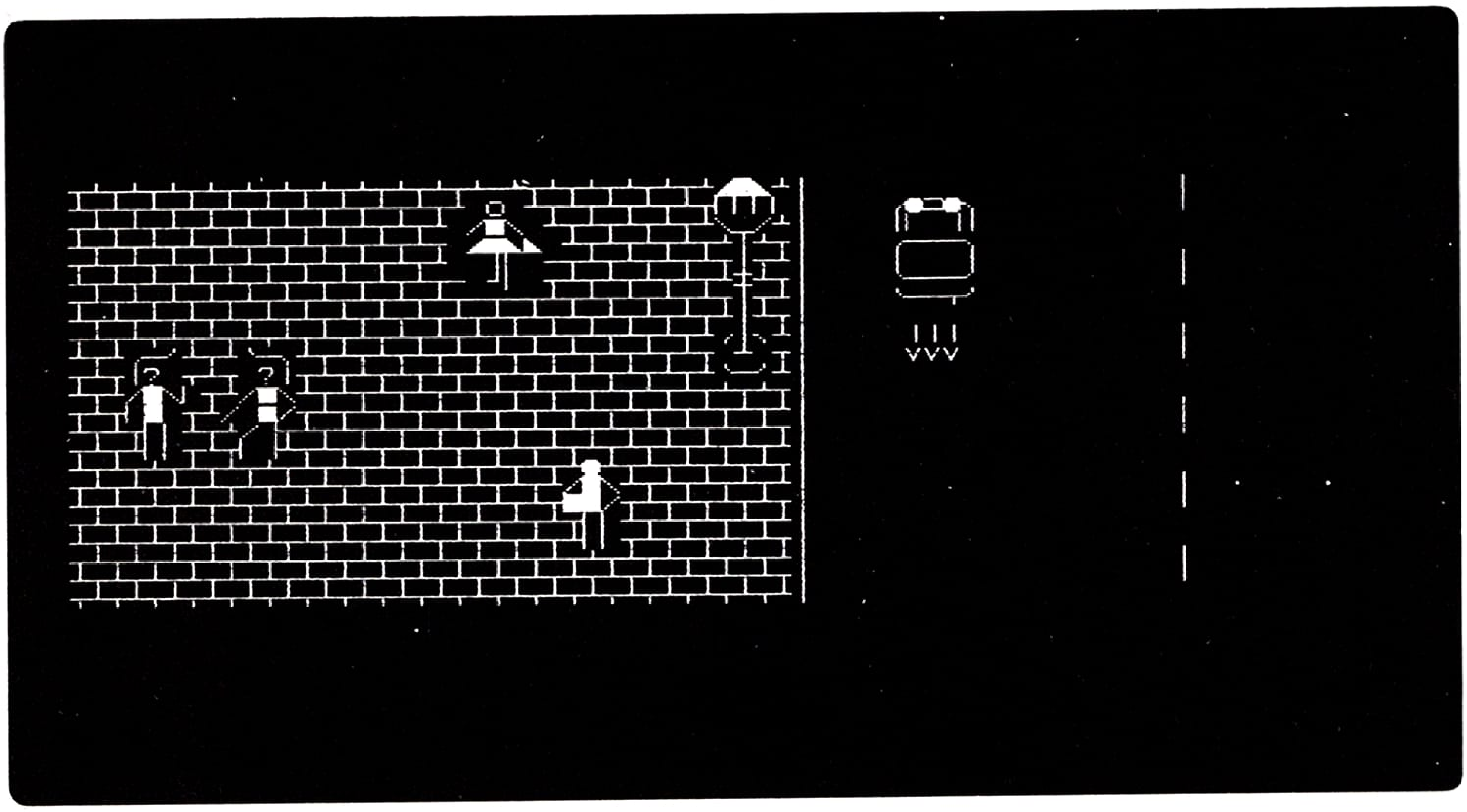


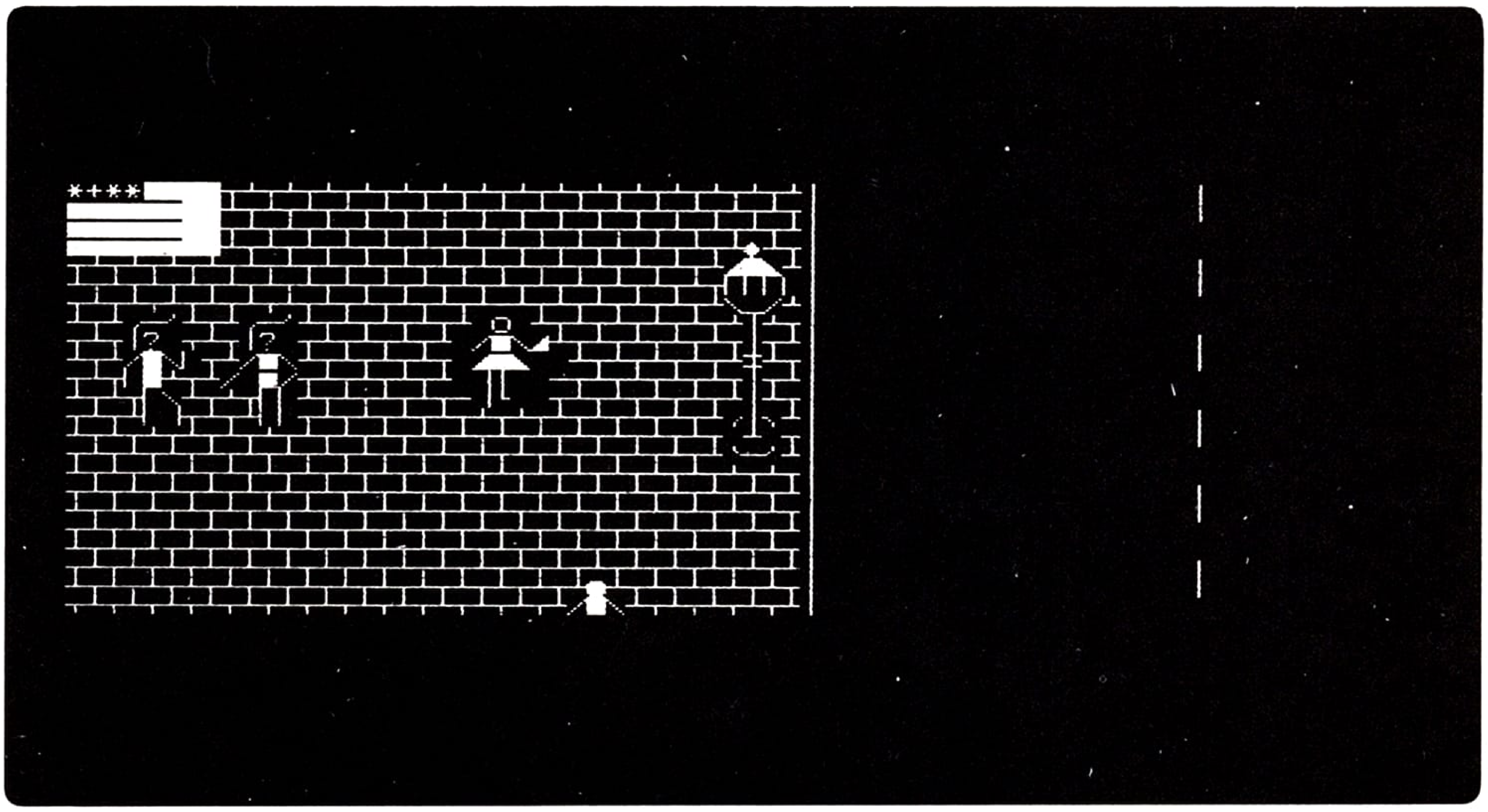
WHISPER

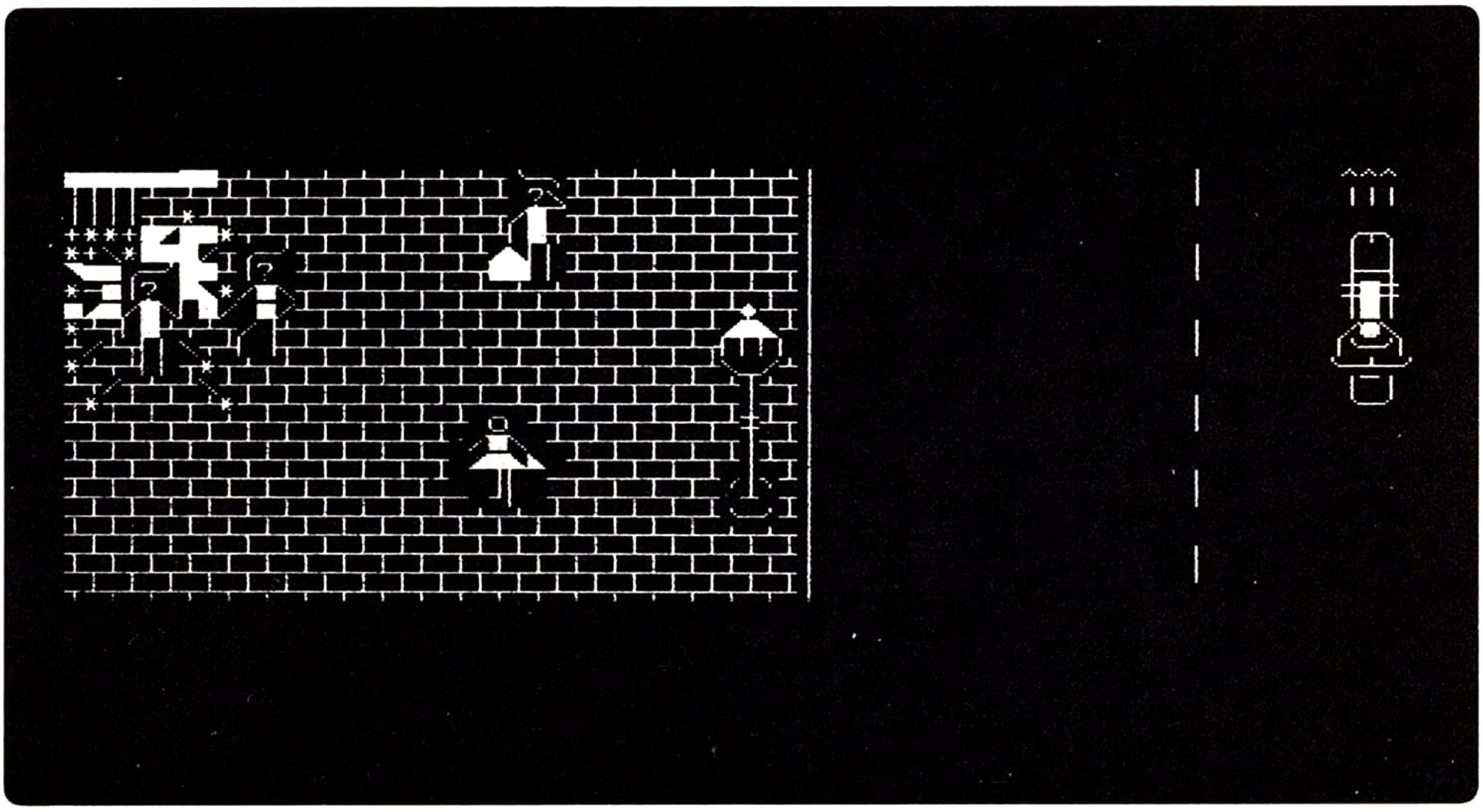
この辺で少し休憩にしたいと思います。

★ある日、KOBEで★









2·8　LD命令(III)

さて、今度は、

ＬＤ　(ｎｎ), Ａ

LD (アドレス), A

という命令です。

これは、ちょうど先程の「ＬＤ　Ａ, (アドレス)」の逆さまのようですね。

ここで指定されているメモリへ、Ａレジスタの内容を代入するという命令なのです。

メモリにＡレジスタの中身を書き込むわけですから、ＢＡＳＩＣで言うと、「ＰＯＫＥ」に相当します。

ＬＤ　(ＨＬ), ｒ

LD (HL), レジスタ

という命令もあり、これもほぼ同等の働きをします。

ＨＬレジスタペアで指定されたアドレスへ、レジスタの「内容」を書き込むのです。

ＢＡＳＩＣにたとえてみれば、

```
LD   (B010),A        --->      POKE &HB010,A
LD   (HL),A          --->      POKE HL,A
```

というところです。

ＬＤ　Ａ, (ｎｎ) はＰＥＥＫ、
ＬＤ　(ｎｎ), Ａ はＰＯＫＥと同じ働きをするのです。

『(』『)』のある位置に注意して下さい。
『,』の後ろから『,』の前へロードするわけです。

★ for example ★

では、Ｂ０１０番地に書き込まれているＡＡを読み出してＢ０１８番地へ書き込む、というプログラムを作ってみましょう。

こうやって

アドレス	マシン語	ラベル	ニーモニック	コメント
			ORG　B000	
B000	3A　10　B0		LD　A←(B010)	読み出して……
B003	32　18　B0		LD　(B018)←A	書き込む。
B006	FF		RST　38H	モニタに戻る。

こうやって

```
B000 3A 10 B0 32 18 B0 FF      : -2 -
```

こうやると

```
h]gb000 ⏎
```

こうなります。

```
B010 AA FF 00 FF 00 FF 00 FF  I
B018 AA FF 00 FF 00 FF 00 FF  I
```

また、ＨＬレジスタペアを利用しても、

アドレス	マシン語	ラベル	ニーモニック	コメント
			ORG　B000	
B000	21　10　B0		LD　HL←B010	B010番地
B003	7E		LD　A←(HL)	から読み出し……
B004	21　18　B0		LD　HL←B018	B018番地
B007	77		LD　(HL)←A	へ、書き込む。
B008	FF		RST　38H	モニターに戻る。

作れます。

```
B010 AA FF 00 FF 00 FF 00 FF  I
B018 AA FF 00 FF 00 FF 00 FF  I
```

16ビットロード

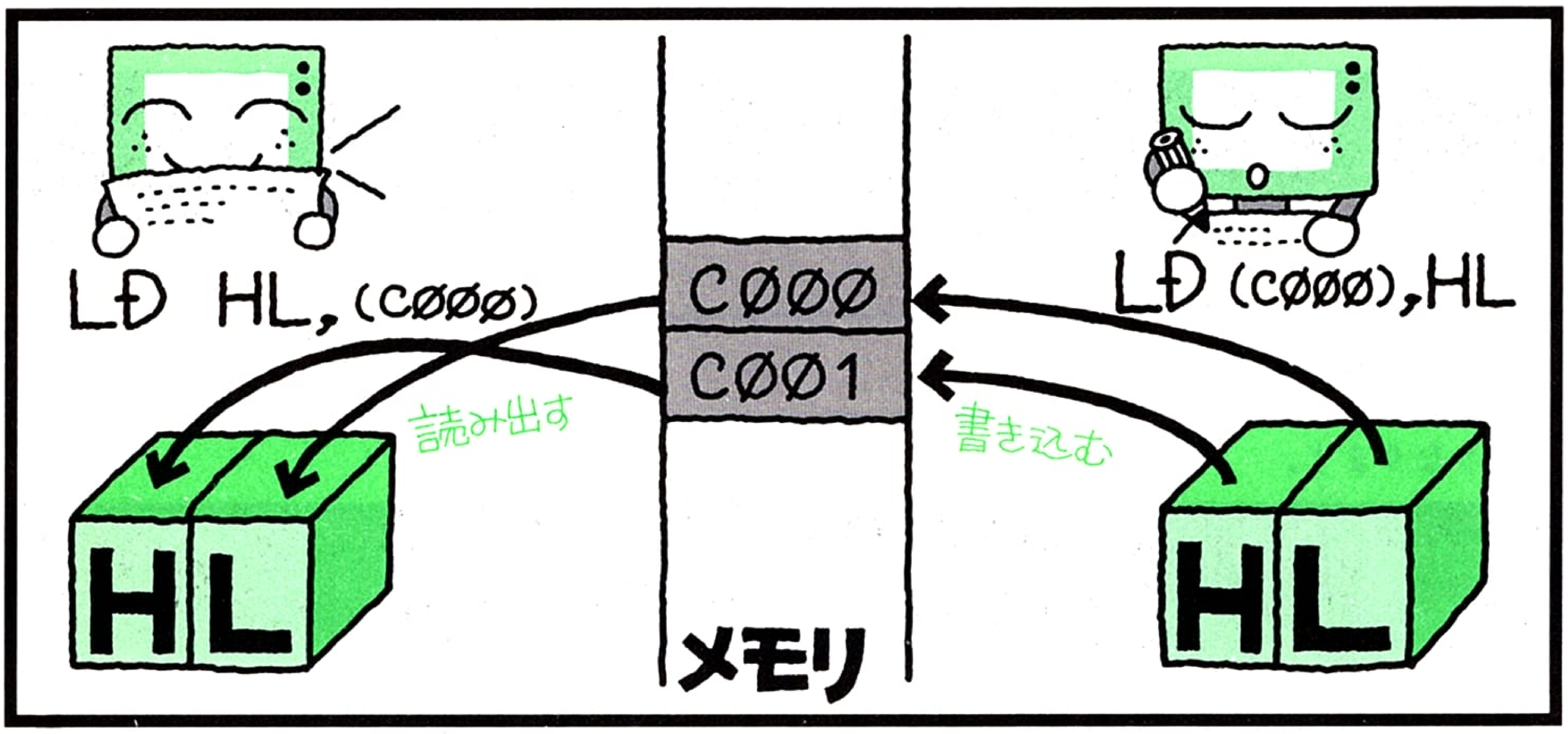
当然のことながら、これにも16ビットロードはあります。

ＬＤ　(ｎｎ), ｄｄ　　**LD (アドレス), レジスタペア**

です。

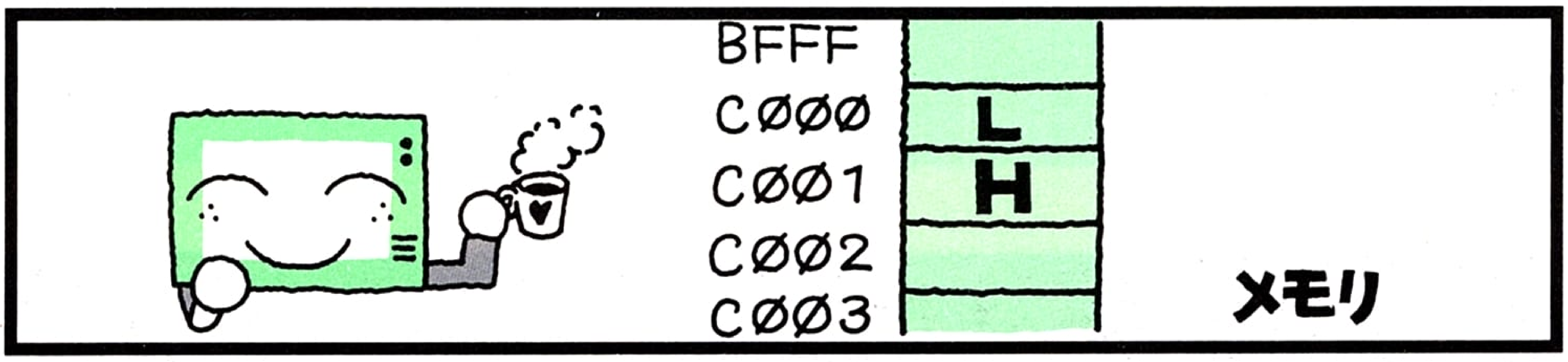
レジスタペアの内容を、指定されたアドレスとその次のアドレスへ書き込む命令です。

少しわかりやすいように、ＬＤ　ｄｄ,(ｎｎ) と比較しながら図解してみましょう。

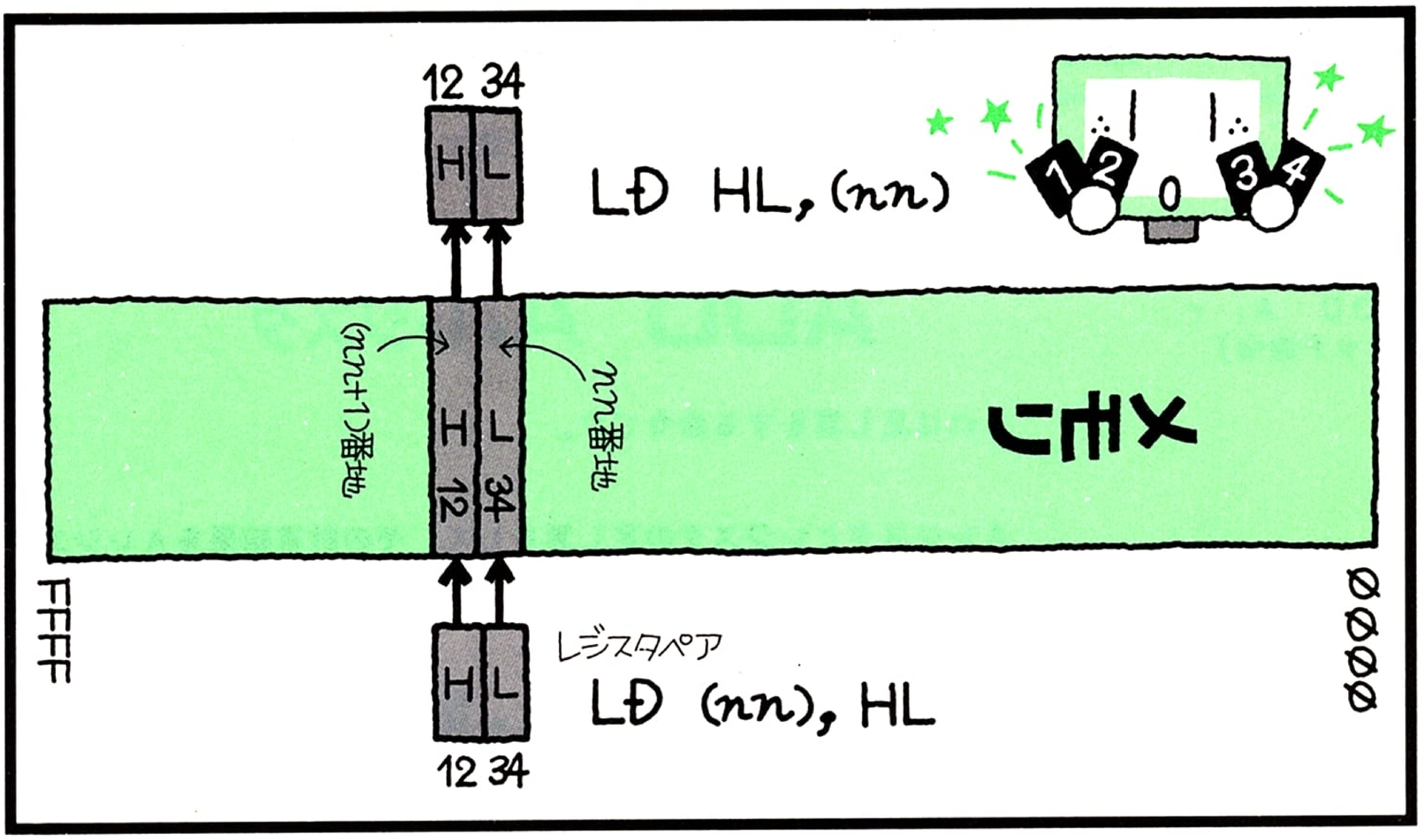


レジスタペアの、後ろのレジスタ（ＨＬレジスタの場合ならＬ）が指定されたアドレスへ、前のレジスタ（この場合Ｈ）がその次のアドレスへ書き込まれるわけですから、ちょうど上の図のようになります。

ですから、レジスタで扱う時はＨＬの順で扱っていてもメモリ上で扱う時は、『ＬＨ』の順で扱うということに気がついて下さい。



もっとわかりやすく言うとしたら、こうなるでしょうか。



たとえば、今「１２３４」という数を扱いたいとします。

ＨＬレジスタに入れて使う時は、**Ｈ**に「**１２**」、**Ｌ**に「**３４**」を、代入して使うわけです。

しかし、メモリにこの数を保存しておく時は、**若いアドレスから、「３４」「１２」と保存する**わけです。

こうしておいてはじめて、この「３４１２」とメモリに保存されている数をＨＬレジスタに読み出したときに、ちゃんと「１２３４」の順で入ることになります。

ただし、レジスタペアに入っている数を一時メモリに書き込んでおいて、あとから読み出すという場合は、入れ換えなくてかまいません。
なぜなら、レジスタペアをひっくり返して保存した数値は、読み出す時再びひっくり返されるので、結局もとにもどってしまうからです。

2·9 演算命令

電子計算機なのですから、当然計算する命令はあります。
まずは、

ADD A, r
(アッド命令)

ADD A,レジスタ

これは**足し算をする命令**です。

Aレジスタとレジスタの足し算をして、その計算結果をAレジスタに代入するのです。

ADD A, n

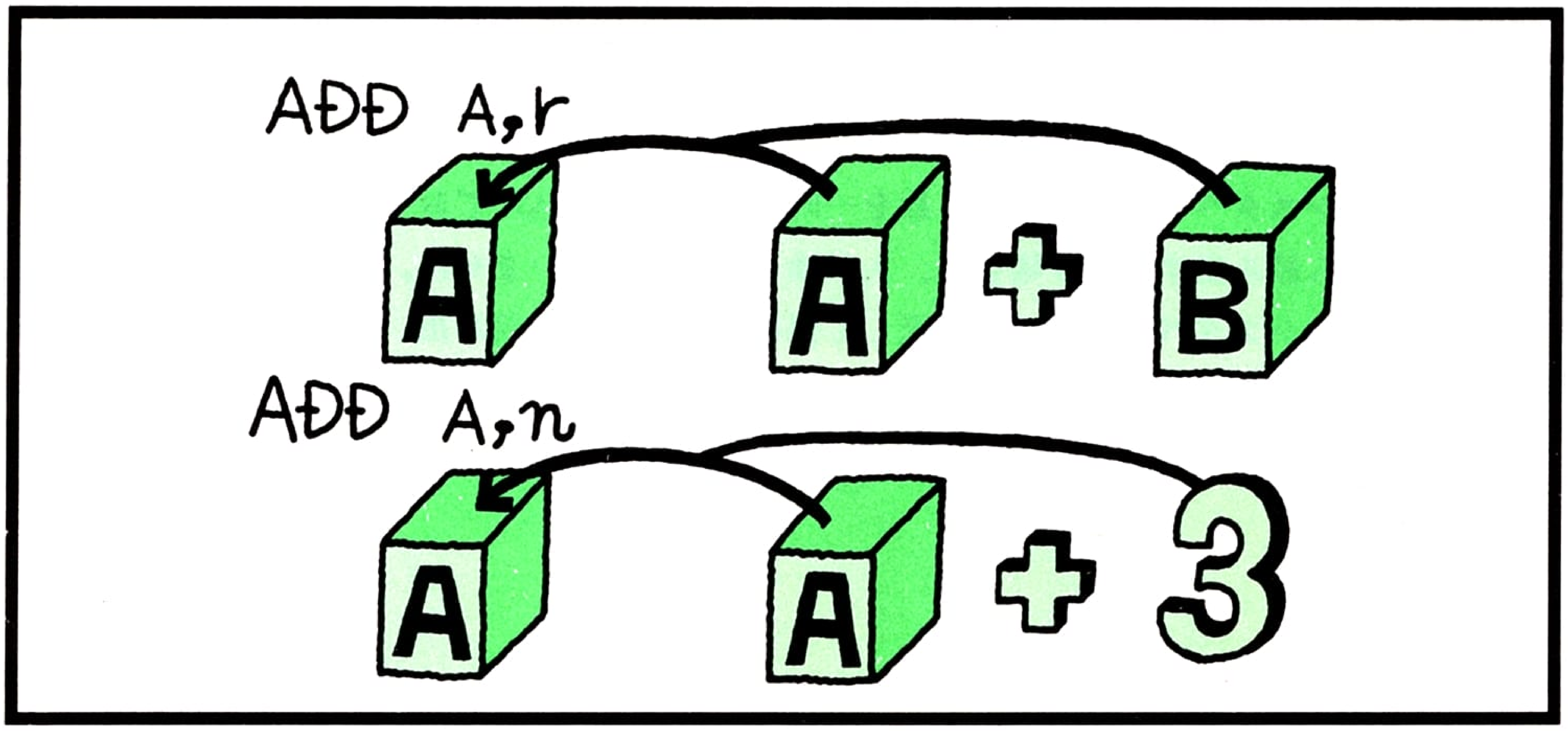
ADD A,数値

という命令もあります。
この方が直感的に理解しやすいですね。**Aレジスタに直接数値を足して、その結果をAレジスタに代入する**のです。

いつもの通りＢＡＳＩＣでたとえると、

```
ADD   A,B    --->    A=A+B
ADD   A,3    --->    A=A+3
```

となりますが、「ＡＤＤ」という単語自体が「足す」という意味ですから、いちいちＢＡＳＩＣに訳す必要もないようですね。



いまのＡＤＤ命令は、Ａレジスタだけが関係する命令でした。
Ｂやでやもやレジスタ、それに、ほかのレジスタではこのような足し算はできません。

ですが、プログラムを作っていく上でちょっとした足し算をするような場合はいくらでも出てきます。例えば､「ＦＯＲ～ＮＥＸＴ」のようなステートメントを実行させる時、変数を「＋１」ずつしなければなりません。
このような場合、いちいちＡレジスタへ代入してから足し算を行い、またもとのレジスタに戻す、などとしていたのでは手間がかかりすぎてしまいます。
しかしご安心ください。一番良く使う「＋１」という足し算だけはどのレジスタでもできるようになっているのです。

INC レジスタ

ＩＮＣ　ｒ
（インクリメント）

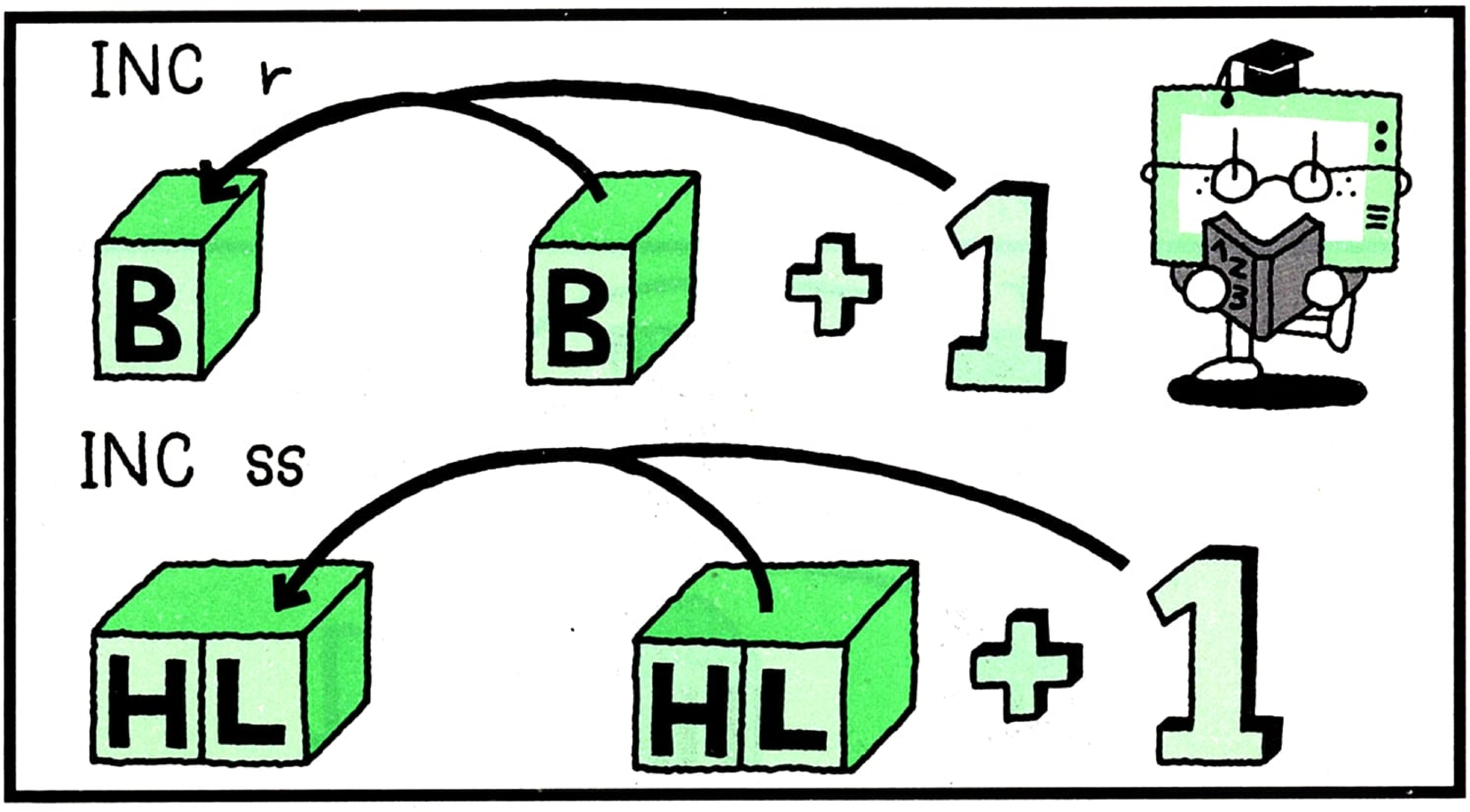
です。
これは、**レジスタの中身に１を加えます。**

また、レジスタペアの場合でも、

INC レジスタペア

ＩＮＣ　ｓｓ

とすることによって、**４ケタの16進数に１を加える**ことが可能です。



減算命令

次は、引き算です。

SUB r
(サブトラクト)

SUB レジスタ

という命令があります。

これは、**Aレジスタから指定されたレジスタの中身を引き、その結果をAレジスタに代入する**というものです。

SUB n

SUB 数値

今度は、**Aレジスタから直接指定された数値を引く**、というものです。

どちらの場合も、Aレジスタから引くということを、命令を書く時に指定しません。それは、このようなことができるのはAレジスタのみだと、わかり切っているためです。

このように、引き算ができるのは原則としてAレジスタだけですが、ADDと同じように「－1」という計算だけはどのレジスタでも行えるようになっています。

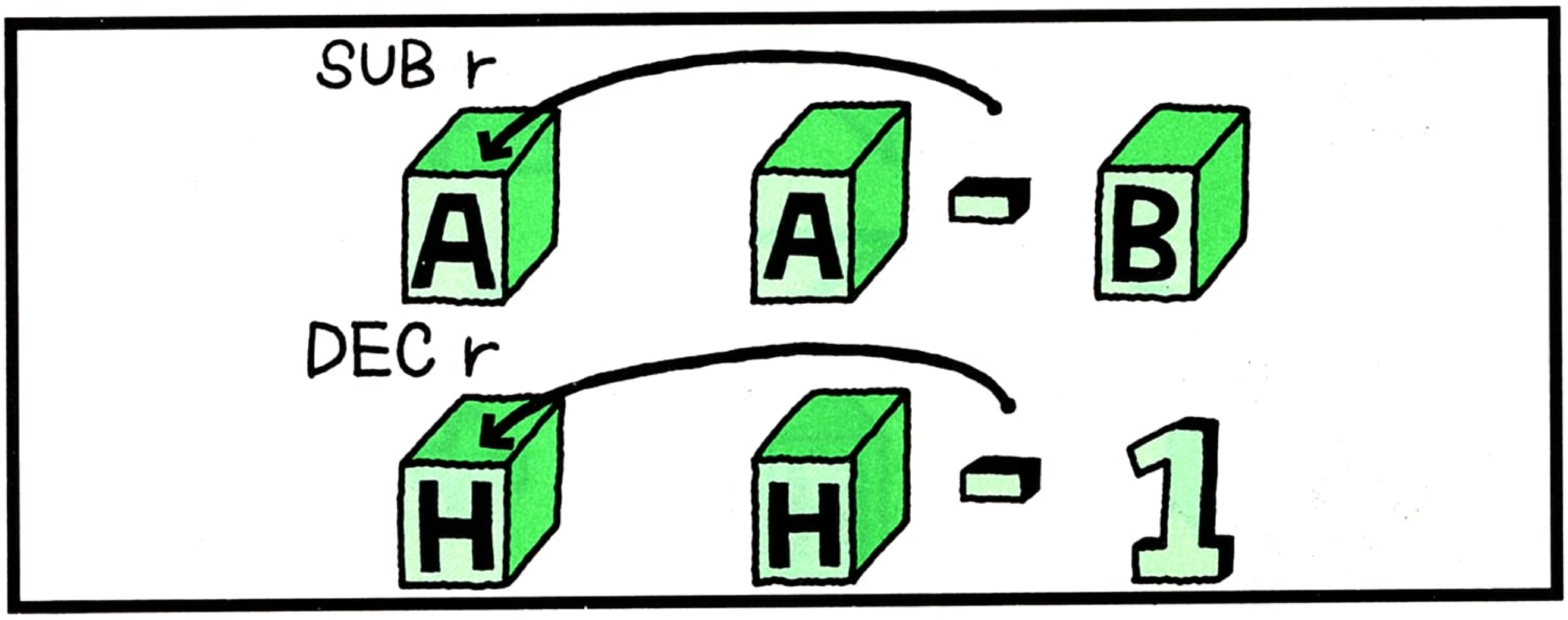
DEC r
(デクリメント)

DEC レジスタ

DEC ss

DEC レジスタペア

です。



★ focus ★

マシン語でできる計算はこれだけです。

かけ算やわり算をする命令はマシン語にはないのです。

したがって、かけ算やわり算をしたい時はそのプログラムを作らなければなりません。

しかし、あまり使うこともないですから、必要になった時に作ればいいでしょう。

★ look at ★

ＡＤＤやＳＵＢ命令の中に「**ADD　A，A**」や「**SUB　A**」というのがあります。

一見普通の命令ですが、実は特殊なことができるのです。

ADD A, A

ADD　A，A

Aレジスタに A レジスタの中身を足す命令です。

つまり、**Aレジスタの中身を「２倍」する**命令なのです。

１回使うだけで、「２倍」にすることができますから、２回使うと「4倍」に、３回で「８倍」にすることができます。

SUB A

SUB　A

AレジスタからAレジスタの中身を引く命令です。

つまり、Ａレジスタの中身がどんな値であったにせよ同じ値を引くわけですから、**結果は「0」**になります。

Ａレジスタを「０」にセットする命令なのです。

2・10　JP命令

ＢＡＳＩＣで「ＧＯＴＯ」や「ＩＦ～ＴＨＥＮ～」にあたる命令です。

ＪＰ　ｎｎ
（ジャンプ）

JP アドレス

この命令は「ＧＯＴＯ」と同じ働きをします。

Ｂ０２０番地に飛びたい時は、「ＪＰ　Ｂ０２０」となり、マシン語では「**Ｃ３**」がＪＰ命令を表わしますので「Ｃ３　２０　Ｂ０」となります。

ＪＰ　Ｘ，ｎｎ

JP 条件，アドレス

この命令は「ＩＦ～ＴＨＥＮ～」と同じ働きをします。
指示された条件によって飛ぶか飛ばないかを判断するのです。

しかし、この条件の出し方がＢＡＳＩＣの時とは違っているので気をつけなければなりません。
ＢＡＳＩＣの時は、

```
IF A=777 THEN PRINT"BONUS"
```

や

```
IF H$="♥" THEN S=S+100
```

などと、「等号」や「不等号」を使って変数を比較し、そしてその式が正しく成り立っている時は「ＴＨＥＮ」以降の処理をしていました。式を見て判断していたのです。

しかしマシン語の場合は、変数の比較と、その判断をわけて行います。
フラグというものを見て判断するからなのです。

フラグ

そうです。**Ｆレジスタ**のことです。

このレジスタは私たちが勝手に書き込むことはできませんが、演算などを行ったりすると（例えばＡＤＤやＳＵＢ命令です）、ＣＰＵが、自動的に「今の演算はこんな状況だった」と書き表わします。

このフラグは私たちが、**Ｘ（イグザミン・レジスタ）**を使って見ることもできますが、ややこしいと思いますので、今は別に理解する必要はありません。

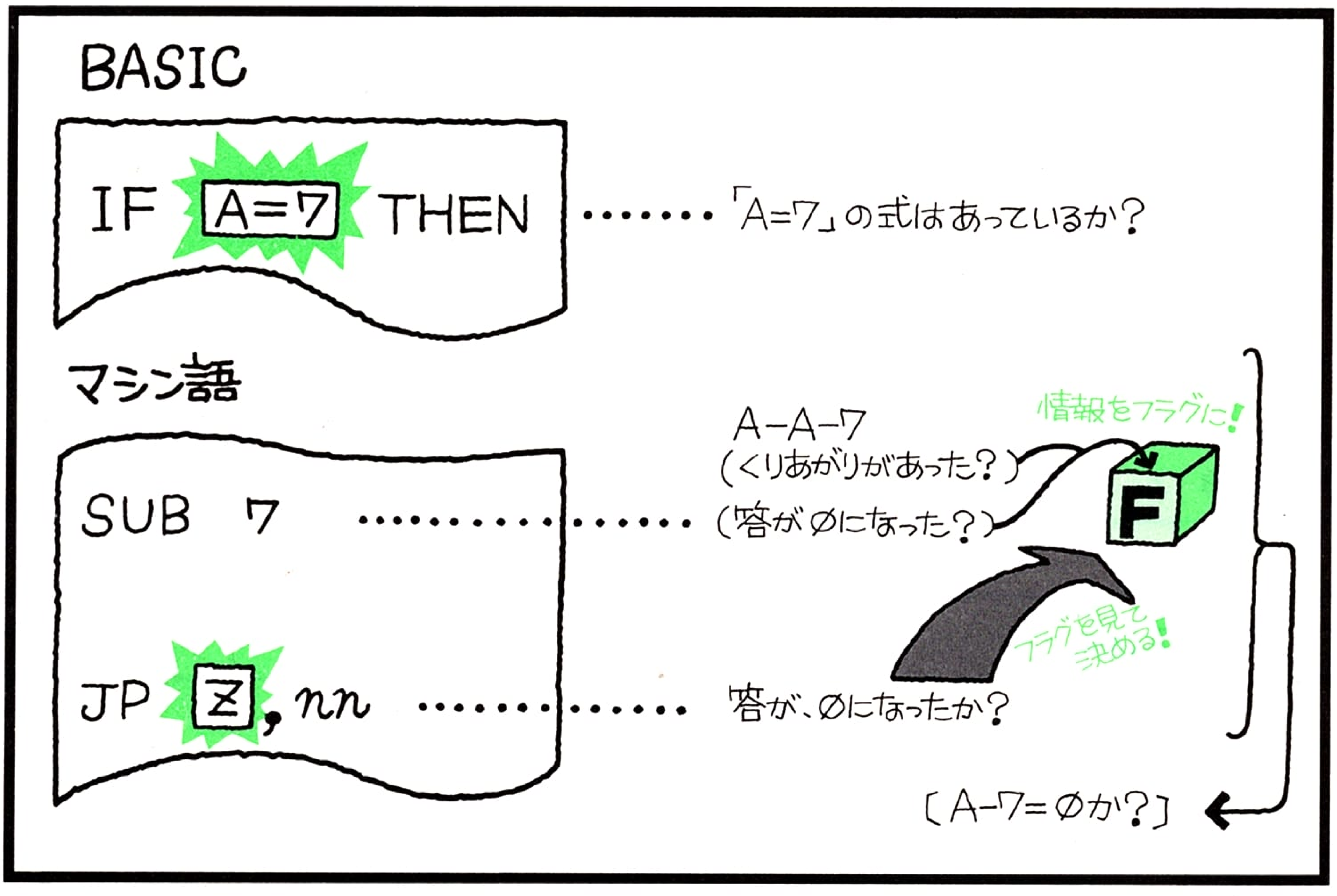
なぜなら、フラグを見て条件判断をするのは『ＣＰＵ』であって、私たちではないからです。

したがって、フラグの中身もＣＰＵにわかりやすいように書かれています。

私たちには、「フラグの中身」を理解するよりも、**「フラグの役目」**を理解する方が大切なのです。

これは実に簡単。

条件判断する時の「鍵」の役目をしているのです。



条件の種類

その条件の中で最もよく使われるのが、

JP　Z，nn
(ジャンプ・イフ・ゼロ)

JP Z，アドレス

です。

「演算結果が0ならばジャンプする」

という命令です。

ほかにも、

JP　NZ，nn
(ジャンプ・イフ・ノット ゼロ)

JP NZ，アドレス

「演算結果が0でなければジャンプする」

JP　C，nn
(ジャンプ・イフ・キャリー)

JP C，アドレス

「演算してくり上がったらジャンプする」

JP　NC，nn
(ジャンプ・イフ・ノット キャリー)

JP NC，アドレス

「演算してくり上がらなかったらジャンプする」

などがあります。

もちろんすべて「フラグ」を見て判断します。

普通よく使う、ＢＡＳＩＣの

```
IF A=7 THEN
```

などは、マシン語では

```
SUB 7                                   A←A－7
```

と、引き算をしておいてから、

```
JP   Z,nn                               もし答が0ならば……
```

として、フラグを見て判断します。

しかし、この実行をする時、本当に必要なのはフラグの結果だけであって、時にはＡレジスタの中身を変えたくないこともあります。

CP命令

そこで、普通条件判断をする時には、

CP　レジスタ　　CP　r（コンペア）

や、

CP　数値　　CP　n

という命令を使います。

Aレジスタから、仮にレジスタや数値を引くと「フラグ」がどう変化するかを調べるものです。

「SUB　B」などを行うと、AレジスタからBレジスタを引いた状況を「フラグ」に書くと同時に、その計算結果をAレジスタに代入してしまいます。

しかし、**「CP　B」**では、Aレジスタの内容からBレジスタの内容を引いた状況は「フラグ」に書き込まれますが、計算結果はAレジスタに代入されません。

したがってAレジスタの内容は実行前と変わらないのです。

BASICでの「IF～THEN～」を、マシン語では、「比較」と「判断」の2行にわけて命令するだけのことです。

IF	A−7	=0　THEN	nn
	CP　7		
JP		Z	nn

2・11 CALL命令

ＢＡＳＩＣの「ＧＯＳＵＢ」「ＲＥＴＵＲＮ」にあたる命令もあります。

「GOSUB」は、

CALL nn
(コール)

CALL アドレス

です。

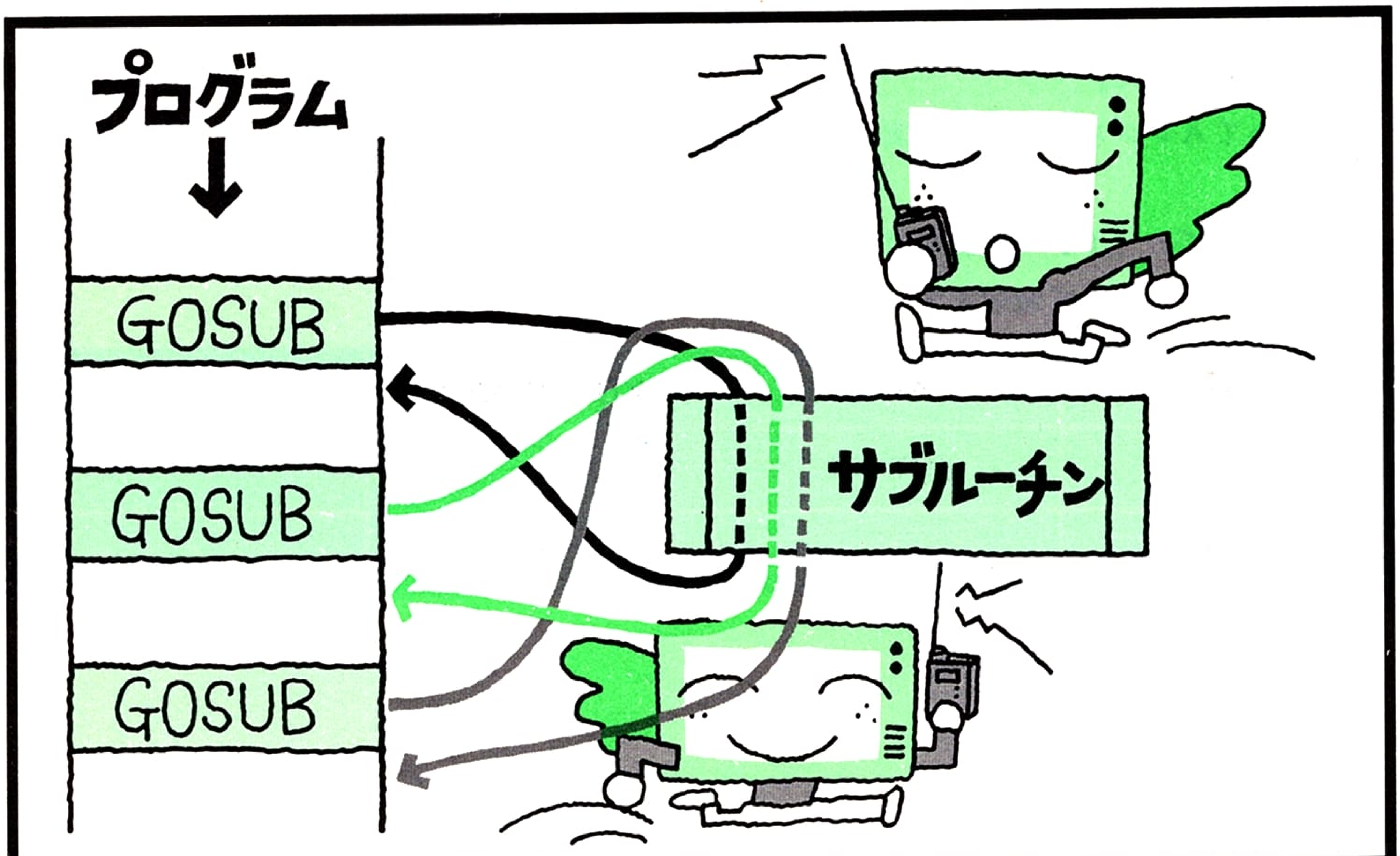
「RETURN」は、

RET
(リターン)

RET

です。

ＢＡＳＩＣと同じように、



などと、何回も１つのサブルーチンを呼ぶためにも使います。

しかし、マシン語の場合そればかりではないのです。

マシン語の場合

ＢＡＳＩＣでは、同じ処理を何回もする時、そのたびに同じプログラムを作っていたのでは能率が悪いというので、わざわざサブルーチンという形にして必要な時に呼び出して使っていました。

ところが、マシン語の場合はたった１回しか使わない処理でもサブルーチンにしてしまうのです。

そして、プログラム自体をサブルーチンのかたまりのようにしてしまうこともあります。

それは、マシン語のプログラムが人間に理解しにくい形になっているからなのです。

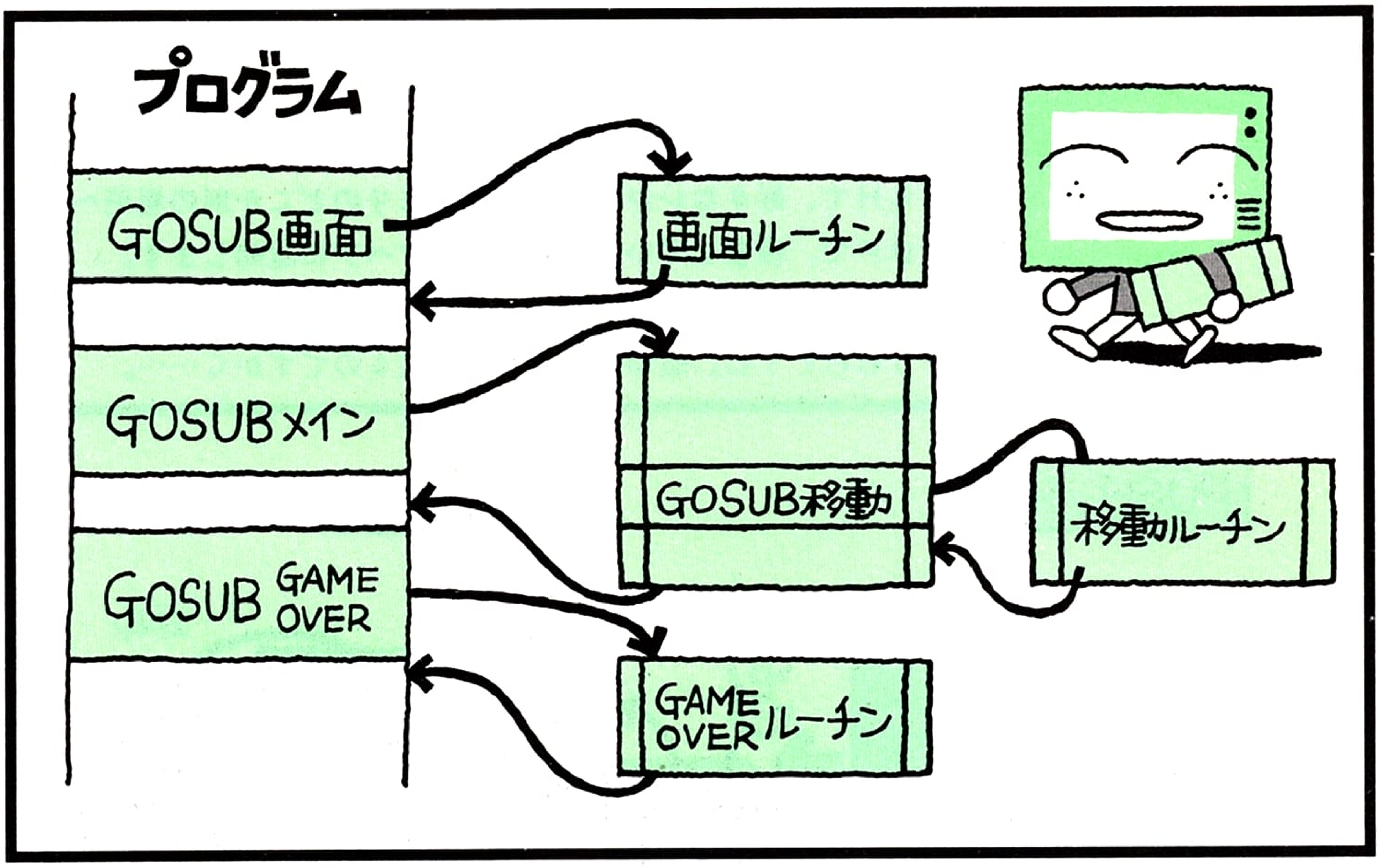
そのため、一目見たぐらいでは全体の流れがつかめません。

ＢＡＳＩＣでは「ＲＥＭ文」があるため、

「このあたりでは、こんなことをしている」

とわかりますが、マシン語ではわからないのです。

そこで、すべてサブルーチンという形にして、



一目でプログラムの流れがわかるようにするのです。

2・12　PUSH命令

今度は、マシン語独特の命令です。

ＰＵＳＨ　ｑｑ
（プッシュ）

PUSH　レジスタペア

と

ＰＯＰ　ｑｑ
（ポップ）

POP　レジスタペア

です。

これは、言ってみれば**「レジスタを一時保管しておく」**という命令なのです。

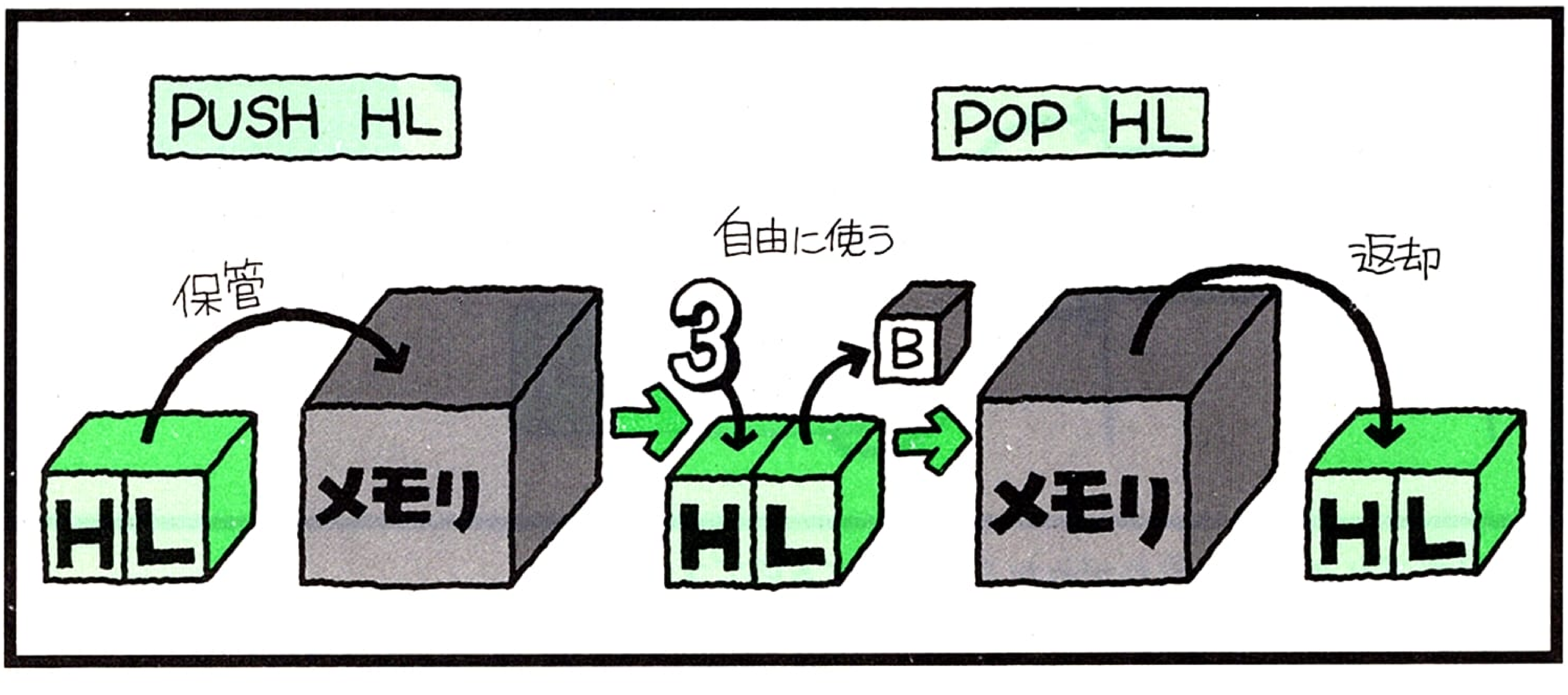
レジスタはたった８つしかありません。ですから８つのレジスタを使ってしまっている時は、もう他のデータは扱えないことになってしまいます。

それでもなお使いたいという時は、今使っているレジスタのデータをいったん別の場所に置いておいてから使い、再び必要になった時レジスタに戻す、ということをしなければなりません。

まさに、このことをやってのけるのが「ＰＵＳＨ」と「ＰＯＰ」なのです。

ＰＵＳＨで、好きなレジスタペアをメモリのどこか別の場所へ保管し、ＰＯＰで、保管していた数をレジスタペアに返却します。

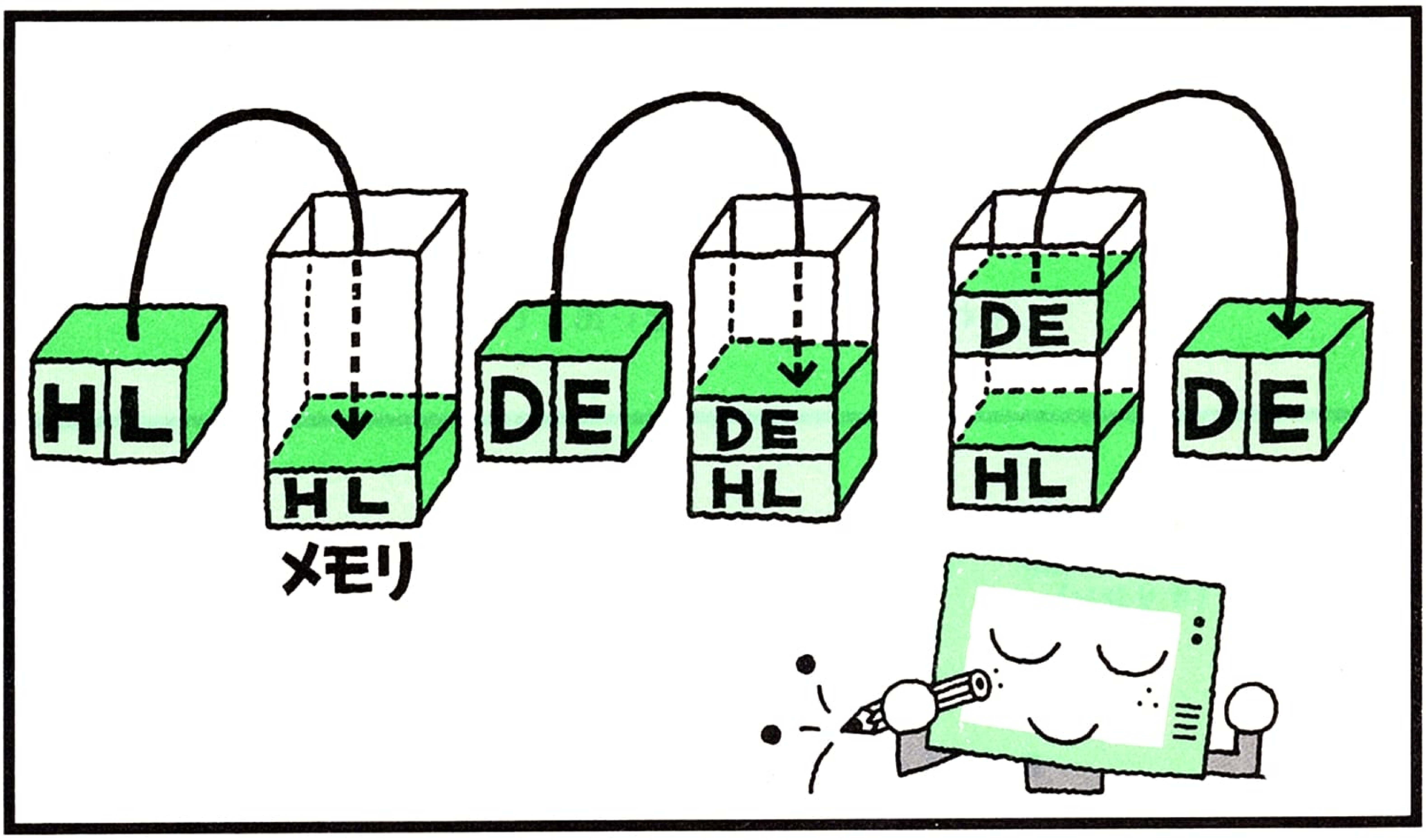
保管している間は、そのレジスタをもうどのように使っても構わないというわけですね。返却したらもとに戻るのですから……。



保管の方法

ＰＵＳＨやＰＯＰは１回しか使えないのではありません。何回でも使えるのです。

そのためメモリにデータを保管する時に一定の決まりがあります。



ちょうど、箱の中にぴったりの大きさの本をつんでいくようにするのです。

いろいろなデータをメモリの中にＰＵＳＨでしまいます。

そして、**ＰＯＰで取り出します。**この時に気をつけて下さい。

メモリの中につまれたデータは上からしか取り出すことができません。したがって、一番新しく保管したデータからしか取り出すことができないのです。

一番古いデータは、一番最後にならないと取り出せません。

ですから、ＰＵＳＨで保管する時は「順番」に注意しましょう。

それと、もう一つ。

保管したデータは必ず同じプログラム中でメモリからすべて取り出しておかなくてはなりません。

預けっぱなしにしておくのはいけないのです。

でないと、怒って暴走するかもしれません。

2・13 まとめ

以上が、マシン語の基本的な命令です。

これだけを使いこなせれば、もう大抵どんなプログラムでも作れます。

他の命令は、知っていた方が便利なこともある、程度の命令ばかりなのですから……。

それでは、もう一度、思い出して下さい。

ORG スタートアドレス

（オリジン）

機　能：プログラムの先頭アドレスの覚え書き。
種　類：ORG
BASIC：――――

NOP

（ノン・オペレーション/ノップ）

機　能：何もしない
種　類：NOP
BASIC：REM？

RST 38H

（リスタート）

機　能：モニターモードに戻る（PC−8801のみ！）
種　類：RST　38H
BASIC：END

LD r, X

（ロード）

機　能：レジスタに数値などを代入する
種　類：LD　r, r'　　LD　r, n
BASIC：LET　r＝r'　　LET　r＝n

LD r,(XX)
（ロード）

機　　能：メモリからレジスタに読み込む
種　　類：LD　A,（nn）　LD　r,（HL）
BASIC：A=PEEK（nn）　A=PEEK（HL）

LD (XX), r
（ロード）

機　　能：メモリにレジスタの値を書き込む
種　　類：LD　（nn), A　LD　（HL), r
BASIC：POKE　nn, A　POKE　HL, r

ADD A, X
（アッド）

機　　能：Aレジスタを使って足し算をする
種　　類：ADD　A, r　ADD　A, n
BASIC：A=A+r　A=A+n

INC X
（インクリメント）

機　　能：レジスタに「1」加算する
種　　類：INC　r
BASIC：r=r+1

SUB X
（サブトラクト）

機　　能：Aレジスタを使って引き算をする
種　　類：SUB　r　SUB　n
BASIC：A=A−r　A=A−n

DEC X
（デクリメント）

機　　能：レジスタから「1」減算する
種　　類：DEC　r
BASIC：r=r−1

JP　アドレス
（ジャンプ）

機　　能：指定したアドレスへジャンプする

種　　類：JP　nn

BASIC：GOTO　nn

JP　X，アドレス
（ジャンプ）

機　　能：フラグを見て条件に適うときジャンプする

種　　類：JP　Z，nn　etc.

BASIC：IF　F＝0　THEN　GOTO　nn

CP　X
（コンペア）

機　　能：Aレジスタから仮に引いたときのフラグを調べる

種　　類：CP　r　　CP　n

BASIC：A－r　　A－n

CALL　アドレス　　RET
（コール）　（リターン）

機　　能：サブルーチンを呼ぶ　　戻る

種　　類：CALL　nn　　RET

BASIC：GOSUB　nn　　RETURN

PUSH　レジスタペア
（プッシュ）

機　　能：レジスタペアを一時メモリに保管する

種　　類：PUSH　qq

BASIC：———

POP　レジスタペア
（ポップ）

機　　能：一時保管していたデータをレジスタペアに返却する

種　　類：POP　qq

BASIC：———

第3章

画面に描く

3・1　VRAM

マシン語の命令はＢＡＳＩＣで言うと、

「LET」
「＋」
「－」
「PEEK」
「POKE」
「IF～THEN～」
「GOTO」
「GOSUB」
「RETURN」
「END」

と、これほどの種類があります。

しかし、肝心の

「PRINT」

がありません。

では、マシン語で画面に絵を描くことはできないのでしょうか？

いいえ、ちゃんと絵は描けるのです。

ＢＡＳＩＣをだいぶ知っている人はすぐにピンと来たと思いますが、そうです。

「POKE」

を使うのですね。

「POKE」を使ってV－RAMに書き込んで表示させるわけなのです。

V-RAM

それでは、復習の意味も含めて「Ｖ－ＲＡＭ」について少し説明しておくことにいたしましょう。

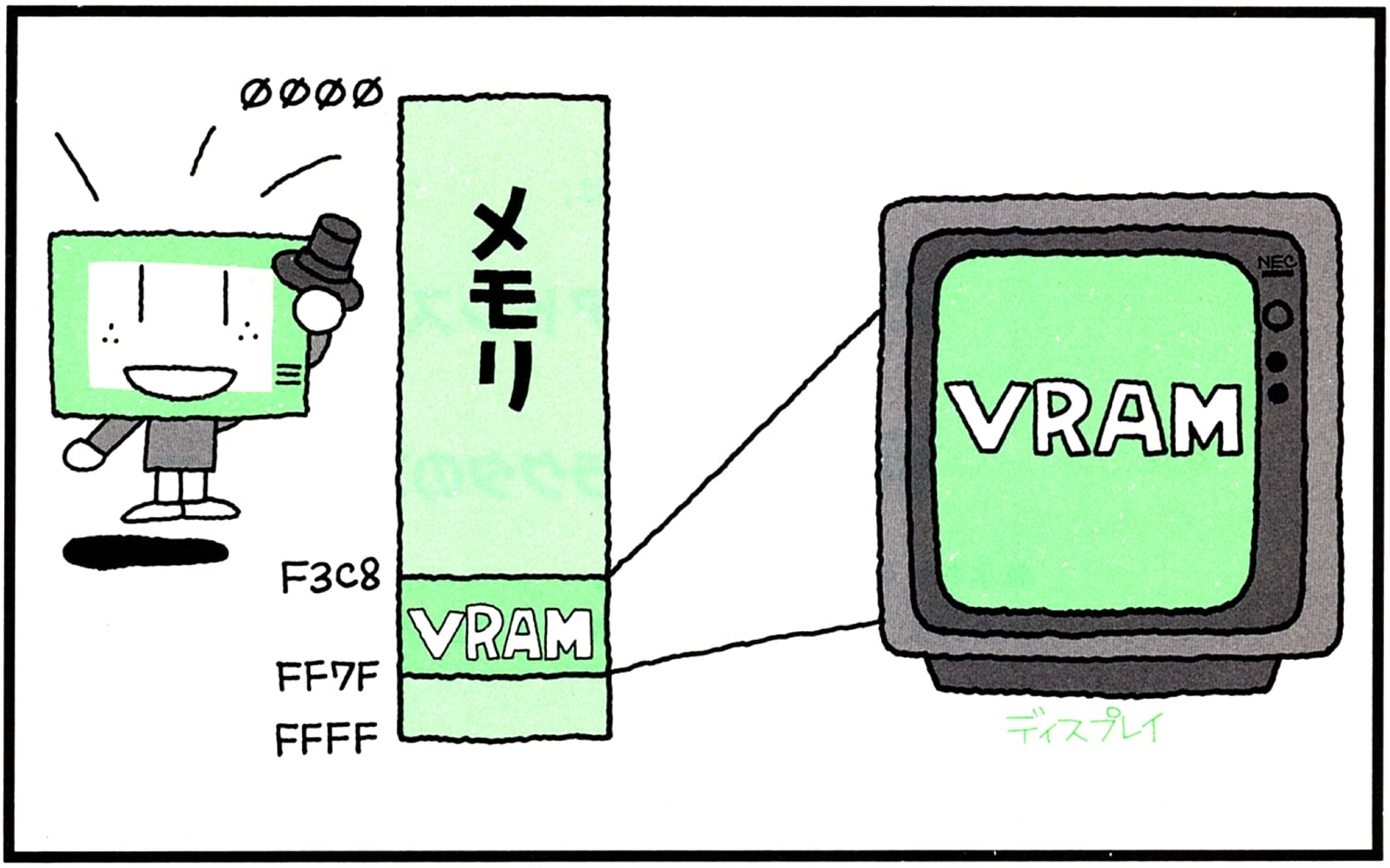
「V－RAM」というのは、メモリの中のある一部分のことを指して言います。

ＰＣ－８８０１の場合なら、F3C8～FF7FがVRAMです。

この領域のメモリだけは特別で、メモリに書き込まれていることがすべて「ディスプレイ」に表示されるのです。

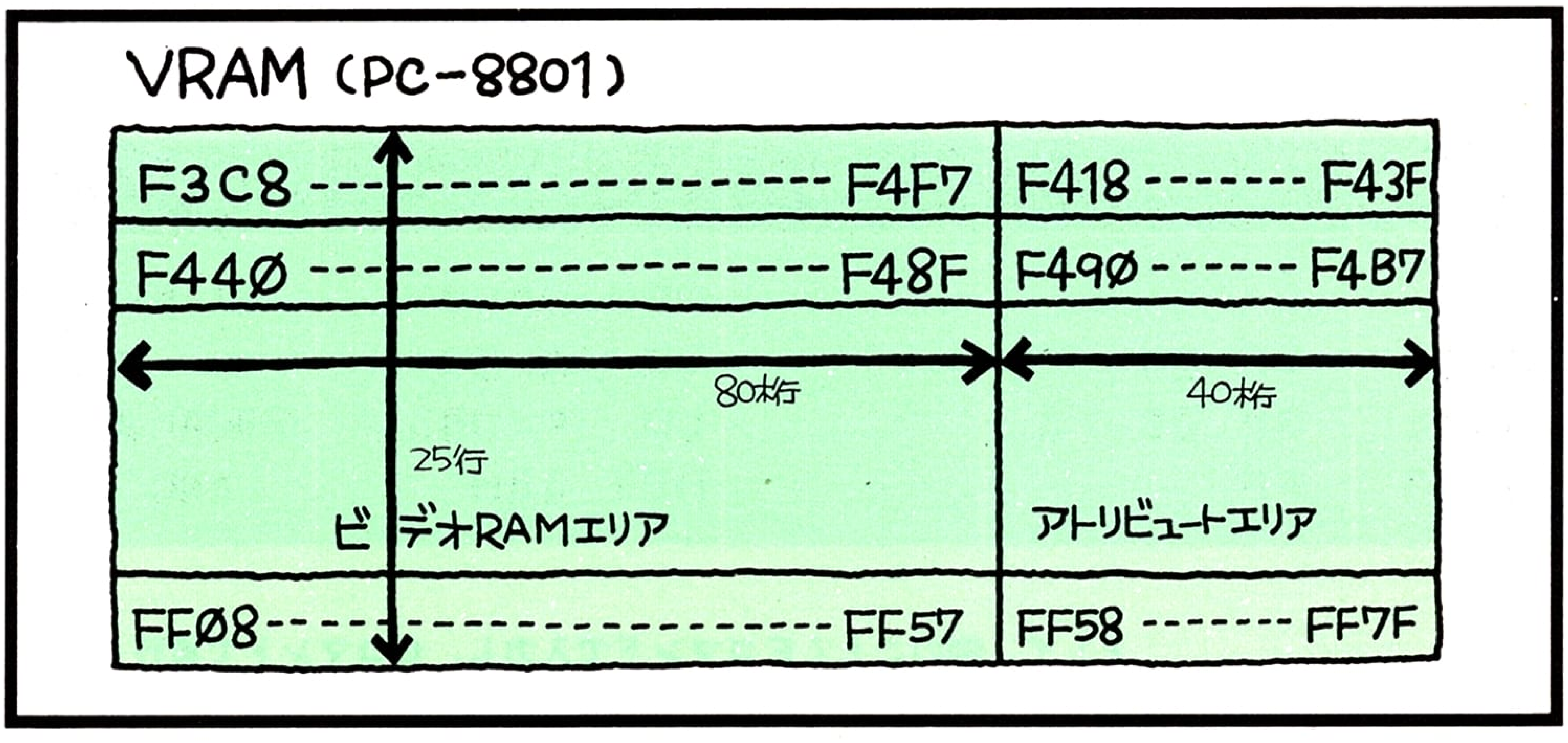
ただし、メモリに書かれている２ケタの16進数が表示されるのではなく、**２ケタの16進数が指しているアスキーコードの文字が表示される**のです。

簡単に言うと「メモリの窓」とでも言えるでしょうか。



ですから、画面に表示したい時には「ＰＯＫＥ」を使ってＶＲＡＭにアスキーコードを書き込めば、その文字がそっくりそのまま画面に表示されるというわけです。

ただし、ＶＲＡＭのどこにでも書き込んでいいというわけではありません。**「アトリビュートエリア」には書かない**ようにします。



3・2 LDでPRINT

では、ともかく、実際に画面に表示させて見ましょう。

まずは一番わかりやすいところから、**左上にハートを表示**させて見ます。

表示させる時に必要なことは、

どこに （アドレス）

何を （キャラクタのアスキーコード）

表示させるかということです。

これは、とても大事なことですのでよく覚えておいて下さい。

「どこに」

これは、一番左上なのですから、**F3C8番地**です。

「何を」

“♥”のアスキーコードは**E9**です。

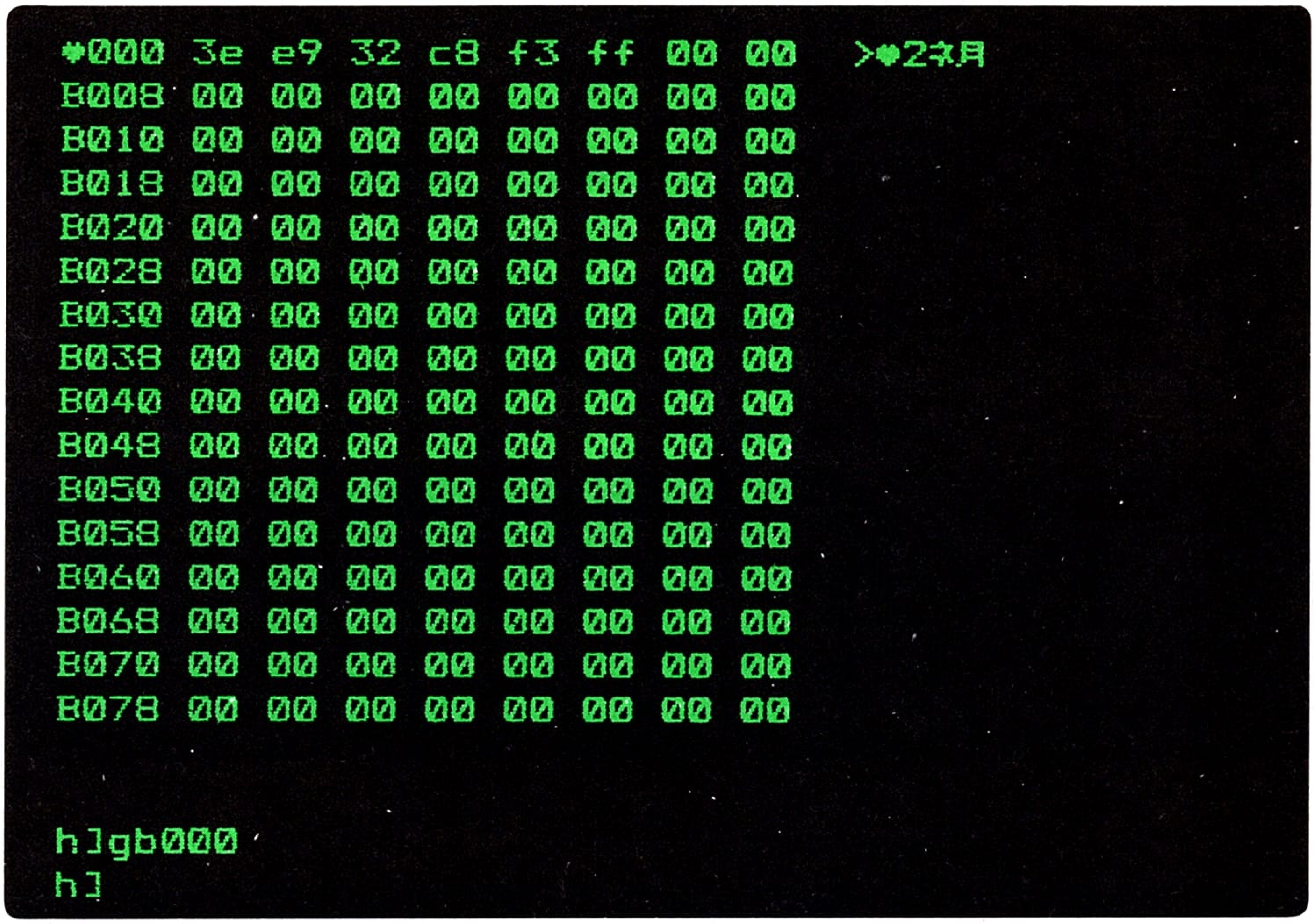
これさえわかれば、後はもう簡単です。

F3C8番地にE9を書き込めばそれだけで画面左上には“♥”が表示されるのです。

F3C8番地にE9を書き込むプログラムを作りましょう。

アドレス	マシン語	ラベル	ニーモニック	コメント
			ORG B000	START
B000	3E E9		LD A, E9	A←E9
B002	32 C8 F3		LD (F3C8), A	(F3C8)←A
B005	FF		RST 38H	END

そして、例のごとく**Eコマンドで入力**し、**Gコマンドで実行**してみるのです。



ちゃんと左上に“♥”が表示されました。

今回は、一番左上だということだったので一発でＦ３Ｃ８番地だとわかりました。

しかし、ＢＡＳＩＣで言う「ＬＯＣＡＴＥ　１０，１０」の位置に表示したい時などはどうしたらいいでしょうか。

アトリビュートエリアに書き込まないように番地を計算しなければなりません。

ややこしそうですね。

下に１つ降りるには、**ビデオＲＡＭエリアの80桁とアトリビュートの40桁を越えて120桁。**

１つ下に降りるごとに120足して、右に１つ移動するごとに１つずつ足してゆく。

しかも、これは16進数で表わさなければなりません。

こんなにややこしいことは、ぜひともコンピュータに計算してもらいたいものです。

そんなわけで、ＶＲＡＭの位置はコンピュータに計算してもらうことにいたしましょう。

私たちは、横にいくつ、縦にいくつの位置かを指定すればいいわけなのです。

これは別にマシン語で作る必要はないのでＢＡＳＩＣで作ります。

40桁モードの時は、

```
100 INPUT "X=";X
110 INPUT "Y=";Y
120 VRAM=(&HF3C8+X*2+Y*120)
130 PRINT HEX$(VRAM)
```

80桁モードの時は、

```
100 INPUT "X=";X
110 INPUT "Y=";Y
120 VRAM=(&HF3C8+X+Y*120)
130 PRINT HEX$(VRAM)
```

とすれば計算できるのです。

たとえば**40桁の場合**、ＲＵＮして、

```
run
X=? ■
```

Ｘ＝０、Ｙ＝０とすると、ちゃんと「**Ｆ３Ｃ８**」が出ます。

```
run
X=? 0
Y=? 0
F3C8
Ok
■
```

それでは、Ｘ＝10、Ｙ＝10の時の計算をしてみましょう。

```
run
X=? 10
Y=? 10
F88C
Ok
■
```

F88C番地だということがすぐわかります。

あとは、さっきのプログラムをちょっと変えるだけで

LOCATE 10, 10

の位置へ表示できますね。

アドレス	マシン語	ラベル	ニーモニック	コメント
			ORG B000	START
B000	3E E9		LD A, E9	A←E9
B002	32 8C F8		LD (F88C), A	(F88C)←A
B005	FF		RST 38H	END

```
B000  3E E9 32 8C F8 FF 00 00  >♥2■
B008  00 00 00 00 00 00 00 00
B010  00 00 00 00 00 00 00 00
B018  00 00 00 00 00 00 00 00
B020  00 00 00 00 00 00 00 00
B028  00 00 00 00 00 00 00 00
B030  00 00 00 00 00 00 00 00
B038  00 00 00 00 00 00 00 00
B040  00 00 00 00 00 00 00 00
B048  00 00 00 00 00 00 00 00
B050  00 00♥00 00 00 00 00 00
B058  00 00 00 00 00 00 00 00
B060  00 00 00 00 00 00 00 00
B068  00 00 00 00 00 00 00 00
B070  00 00 00 00 00 00 00 00
B078  00 00 00 00 00 00 00 00


h]gb000
h]
```

3・3　10コをPRINT

今度は、1コだけなんてセコイこと言わずに、10コ並べて表示させるプログラムを作ってみることにします。

10コ並べるぐらいのこと、BASICなら、

```
10 PRINT "♥♥♥♥♥♥♥♥♥♥"
```

と、いとも簡単なことでした。

しかし、何度も言うようにマシン語に「PRINT」文はないのです。BASICの時のように調子よくはいきません。

ですが、

「全然できない」

「難しい」

と言うものでは、決してありません。

表示させる方法を変えるだけのことなのです。

さっきハートを表示させたプログラムを

```
10 LOCATE 10,10
20 PRINT "♥"
```

「POKE」を使って表示させる

```
10 A=&HE9
20 POKE &HF88C,A
```

という方法に変えたのと同じようにです。

だから、今回も方法を変えてみることにするのです。

1つずつしか表示できないのですから、

```
10 FOR X=10 TO 19
20  LOCATE X,10
30  PRINT "♥"
40 NEXT
```

として10コ並べて表示します。

けれども「LOCATE」や「PRINT」はマシン語にはないので「POKE」で表示する方法に変えて、

```
10 B=&HE9
20 HL=&HF88C
30 FOR I=1 TO 10
40  POKE HL,B
50  HL=HL+1*2
60 NEXT
70 END
```

こうしてみます。

「FOR〜NEXT」もまたマシン語にはない命令なので「IF〜THEN〜」に換えてみます。

```
10 B=&HE9
20 HL=&HF88C
30 A=0
40  POKE HL,B
50  HL=HL+1*2
60  A=A+1
70 IF A<>10 THEN 40
80 END
```

こういうプログラムなら、すべてがマシン語に訳せる命令でできているのでそのまま訳すことができます。

```
10 B=&HE9             -> 10 LD  B,E9
20 HL=&HF88C          -> 20 LD  HL,F88C
30 A=0                -> 30 LD  A,0
40  POKE HL,B         -> 40  LD (HL),B
50  HL=HL+1*2         -> 50  INC HL :INC HL
60  A=A+1             -> 60  INC A
70 IF A<>10 THEN 40   -> 70 CP 10:JP NZ 40
80 END                -> 80 RST 38H
```

あとは、アセンブルすればできあがりですね。

アドレス	マシン語	ラベル	ニーモニック	コメント
			ORG B000	
B000	06 E9		LD B, E9	B←E9
B002	21 8C F8		LD HL, F88C	HL←F88C
B005	3E 00		LD A, 0	A←0
B007	70	LOOP	LD (HL), B	POKE (HL), B
B008	23		INC HL	HL←HL+2
B009	23		INC HL	
B00A	3C		INC A	A←A+1
B00B	FE 0A		CP 10	IFA<>10THEN
B00D	C2 07 B0		JP NZ LOOP	
B010	FF		RST 38H	END

となります。

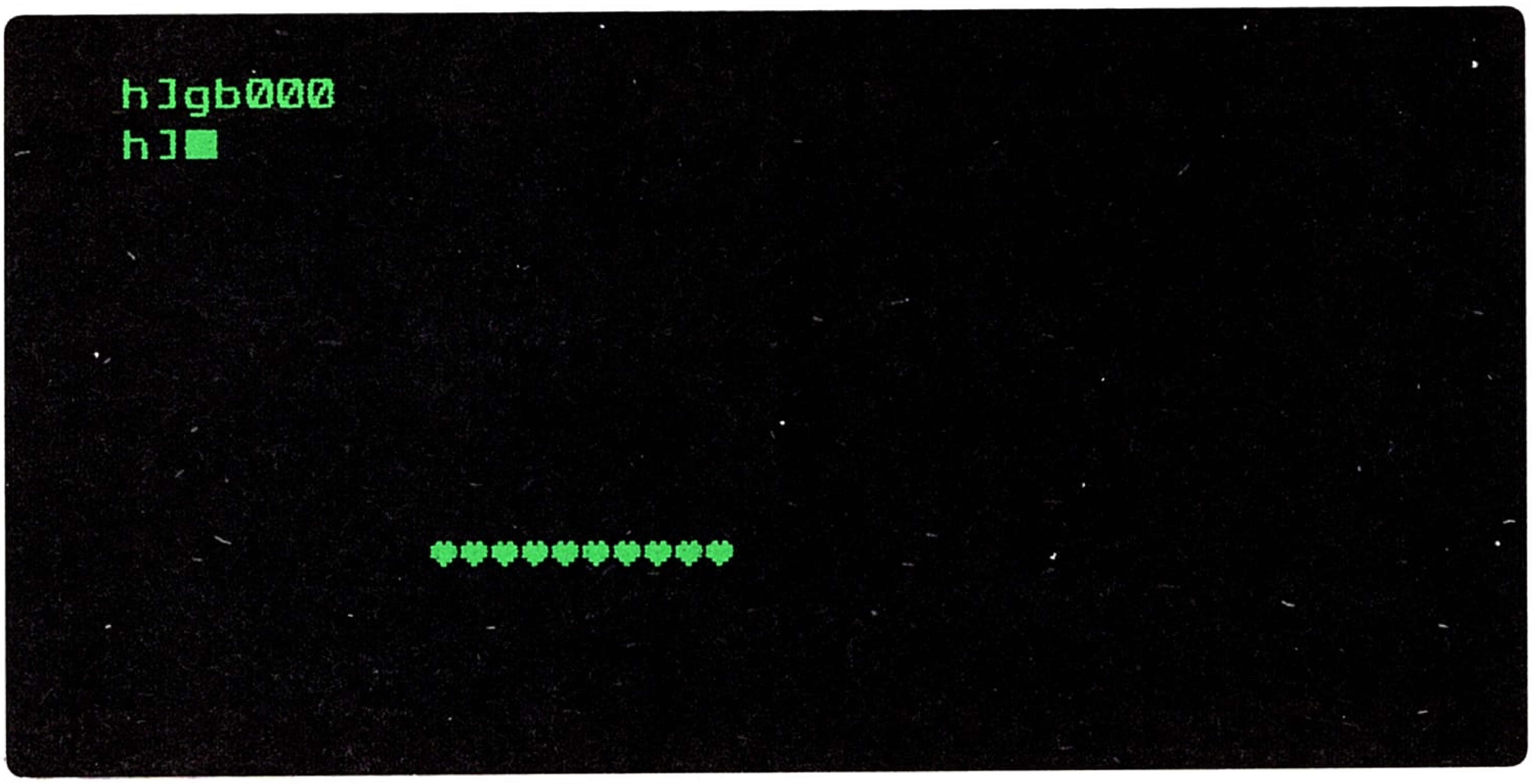
ここで１つ言っておくと、

ラベル

という所に「ＬＯＯＰ」と書かれていますが、これは「N88－ＢＡＳＩＣ」の「ＧＯＴＯ」の時のラベルと同じことです。

「ＪＰ　ＮＺ　ＬＯＯＰ」の時にここへ飛ぶということを示しています。

それでは実行してみましょう。



あれだけ苦労してたったこれだけか、とは思わないで下さい。

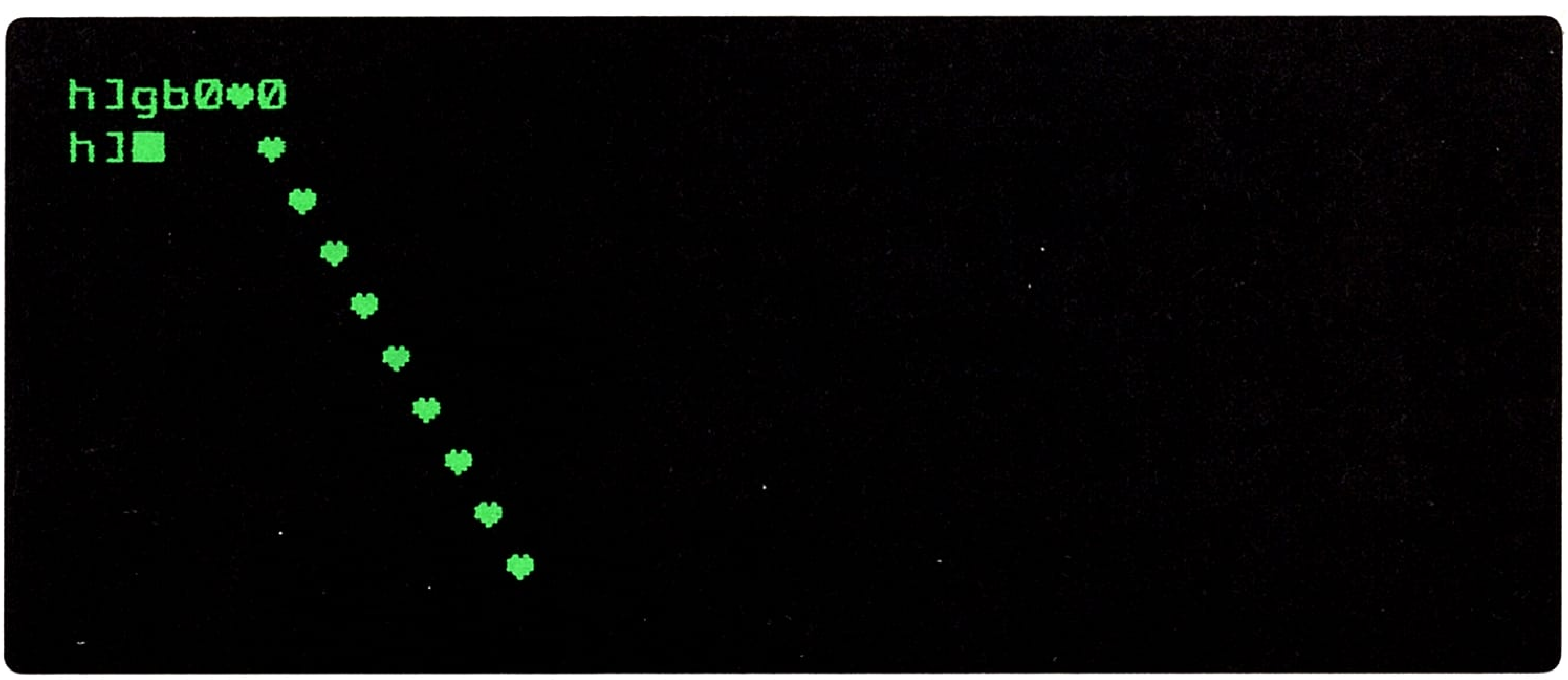
スピードのケタが違うのです。

また、ほとんど同じ考え方で縦に10コ並べることもナナメに10コ並べることも可能です。

```
B000 06 E9 21 D2 F3 3E 00 70
B008 11 78 00 19 3C FE 0A C2
B010 07 B0 FF
```




```
B000 06 E9 21 D2 F3 3E 00 70
B008 11 7A 00 19 3C FE 0A C2
B010 07 B0 FF
```



マシン語は単純な命令しかないので作る時多少手間がかかります。

しかし、単純な命令ばかりでできているので、とても「ゆうずう」がきくのです。

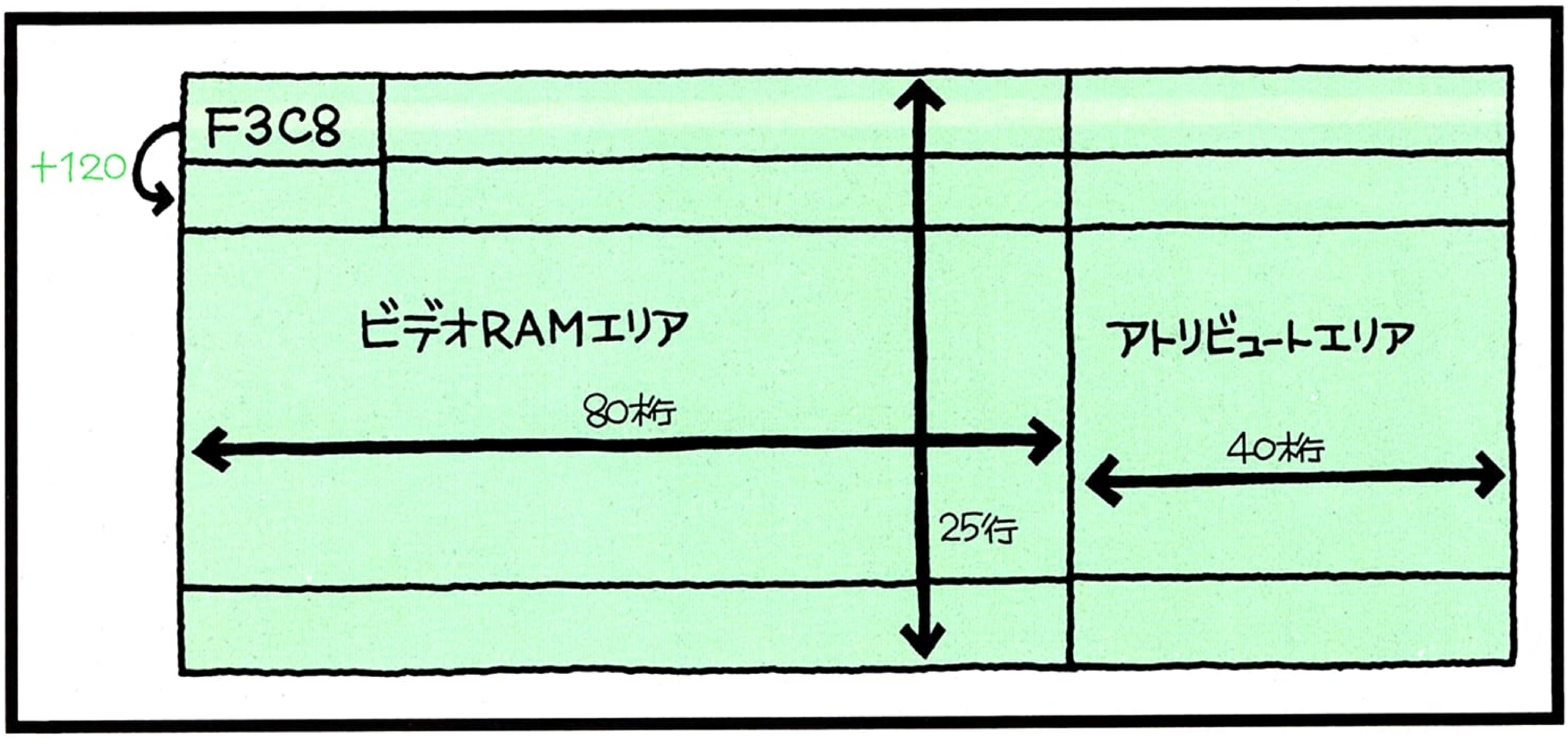
3・4　画面のクリア

では、今度は**画面をクリアするプログラム**を作ってみます。

画面全部をすべて「空白」でうめるのです。

大事なことは、**「どこに」「何を」書き込むか**、です。

ＶＲＡＭはこうなっていました。



出発点が「Ｆ３Ｃ８」で、横に80、縦に25の中をすべてうめつくせば良いわけです。

❶　**まず、Ｆ３Ｃ８から横へ80だけ空白でうめつくします。**
これは、アドレスに「１」足していって80回くり返せばいいわけです。

❷　**そして次の行です。**
行の先頭アドレス（つまり前の行の先頭アドレスに「120」足したもの）を求めてから❶の処理を行います。
（このためには、前の行の先頭アドレスを保管しておく必要があります）

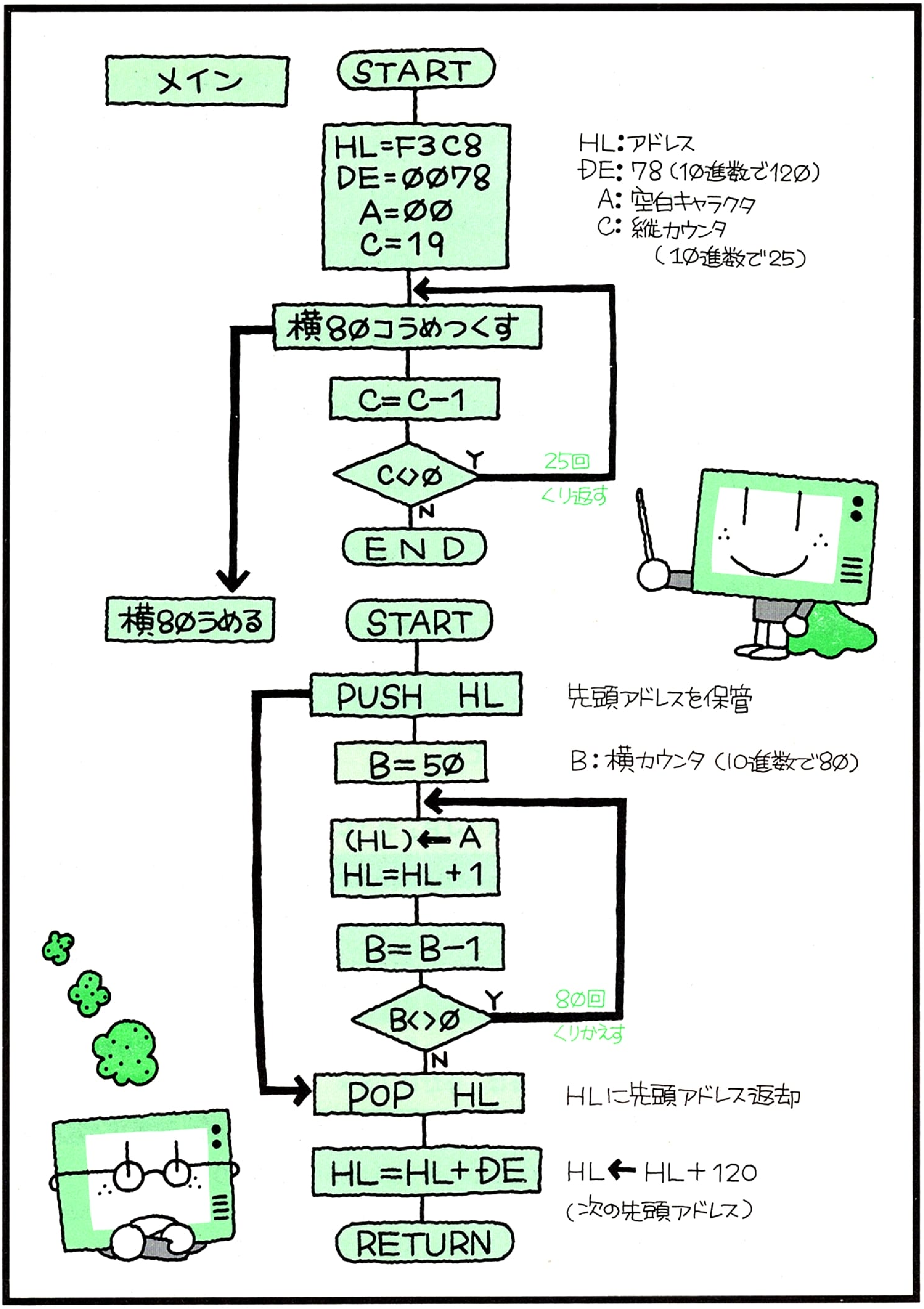
❸　**これを25回くり返します。**

とまあ、これがだいたいの感じです。

あわてずに、１つ１つ落ちついて考えれば何でもないことですから、決して急いだりしないで下さい。

ゆっくり作ったプログラムでも走らせるとものすごく速いのです。

フローチャートで考えると、このようになります。



それでは、これを直接ニーモニックに訳してみましょう。

アドレス	マシン語	ラベル	ニーモニック	コメント
			ORG B000	
B000	21 C8 F3		LD HL, F3C8	HL：アドレス
B003	11 78 00		LD DE, 0078	DE：120
B006	3E 00		LD A, 0	A ：空白キャラクタ
B008	0E 19		LD C, 19	C ：25回
B00A	CD 12 B0	YLOOP	CALL CLR80	
B00D	0D		DEC C	25回
B00E	C2 0A B0		JP NZ YLOOP	
B011	FF		RST 38H	END
B012	E5	CLR80	PUSH HL	
B013	06 50		LD B, 50	B ：80回
B015	77	XLOOP	LD (HL), A	
B016	23		INC HL	80回
B017	05		DEC B	
B018	C2 15 B0		JP NZ XLOOP	
B01B	E1		POP HL	
B01C	19		ADD HL, DE	次の先頭アドレス
B01D	C9		RET	

これでできあがりです。

でも実際にハンドアセンブルしてみると少しつまずいてしまうかも知れません。それは「ＪＰ」や「ＣＡＬＬ」の時です。

アドレス	マシン語	ラベル	ニーモニック	コメント
	CD ?? ??	YLOOP	CALL CLR80	
	0D		DEC C	25回
	C2 ?? ??		JP NZ YLOOP	

まだアドレスをふっていない時は飛び先のアドレスがわかりません。

このような場合は、今だけ「×× ××」と印をつけておきます。

アドレス	マシン語	ラベル	ニッモニック	コメント
	CD ×× ××	YLOOP	CALL CLR80	
	0D		DEC C	25回
	C2 ×× ××		JP NZ YLOOP	

全部訳し終わったら、「×× ××」印も数えてアドレスをふってゆきます。

アドレス	マシン語	ラベル	ニーモニック	コメント
B00A	CD ×× ××	YLOOP	CALL CLR80	←
B00D	0D		DEC C	
B00E	C2 ×× ××		JP NZ YLOOP	25回

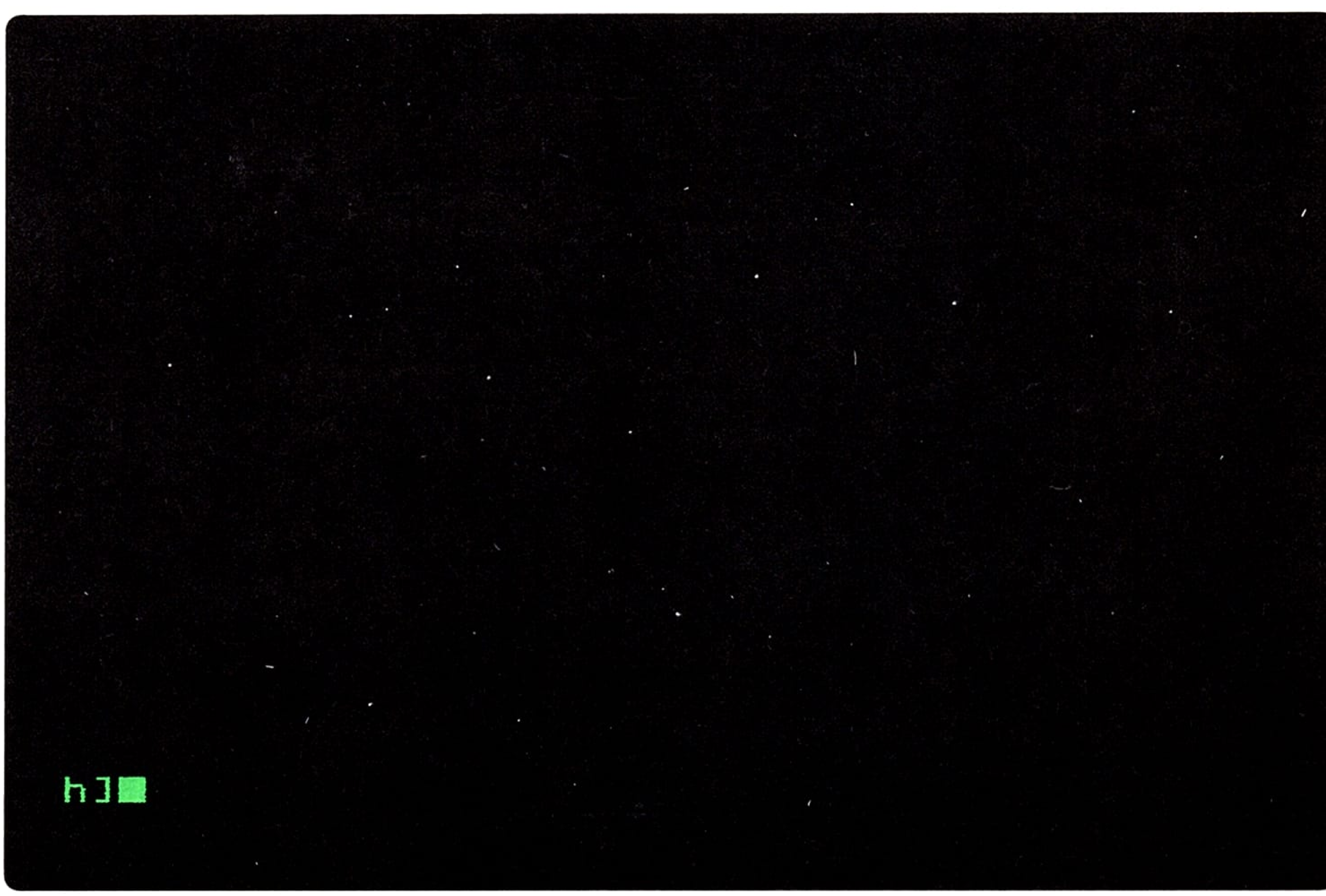
そして、そのあとで**飛び先のアドレスを書き込めばいい**、というわけです。

アドレス	マシン語	ラベル	ニーモニック	コメント
B00A	CD 12 B0	YLOOP	CALL CLR80	←
B00D	0D		DEC C	25回
B00E	C2 0A B0		JP NZ YLOOP	

それでは実行してみましょう。

```
B000 21 C8 F3 11 78 00 3E 00
B008 0E 19 CD 12 B0 0D C2 0A
B010 B0 FF E5 06 50 77 23 05
B018 C2 15 B0 E1 19 C9
```

画面がクリアされます。



3・5　内部ルーチン

でも正直な話、マシン語で画面をクリアするプログラムを作ってもあんまり感動はしないようです。

なぜなら、ＢＡＳＩＣで、

```
PRINT CHR$(12); ⏎
```

や

```
CLS ⏎
```

としても、また

CLR

キーを押しても同じことはできるのですから……。

画面をクリアするなんてことはもともとマイコンにできることだったのであまり感動しなかったというわけです。

しかし、ちょっとだけ注意して考えてみて下さい。マシン語の命令の中に画面をクリアするなんて命令があったでしょうか？

なかったはずです。

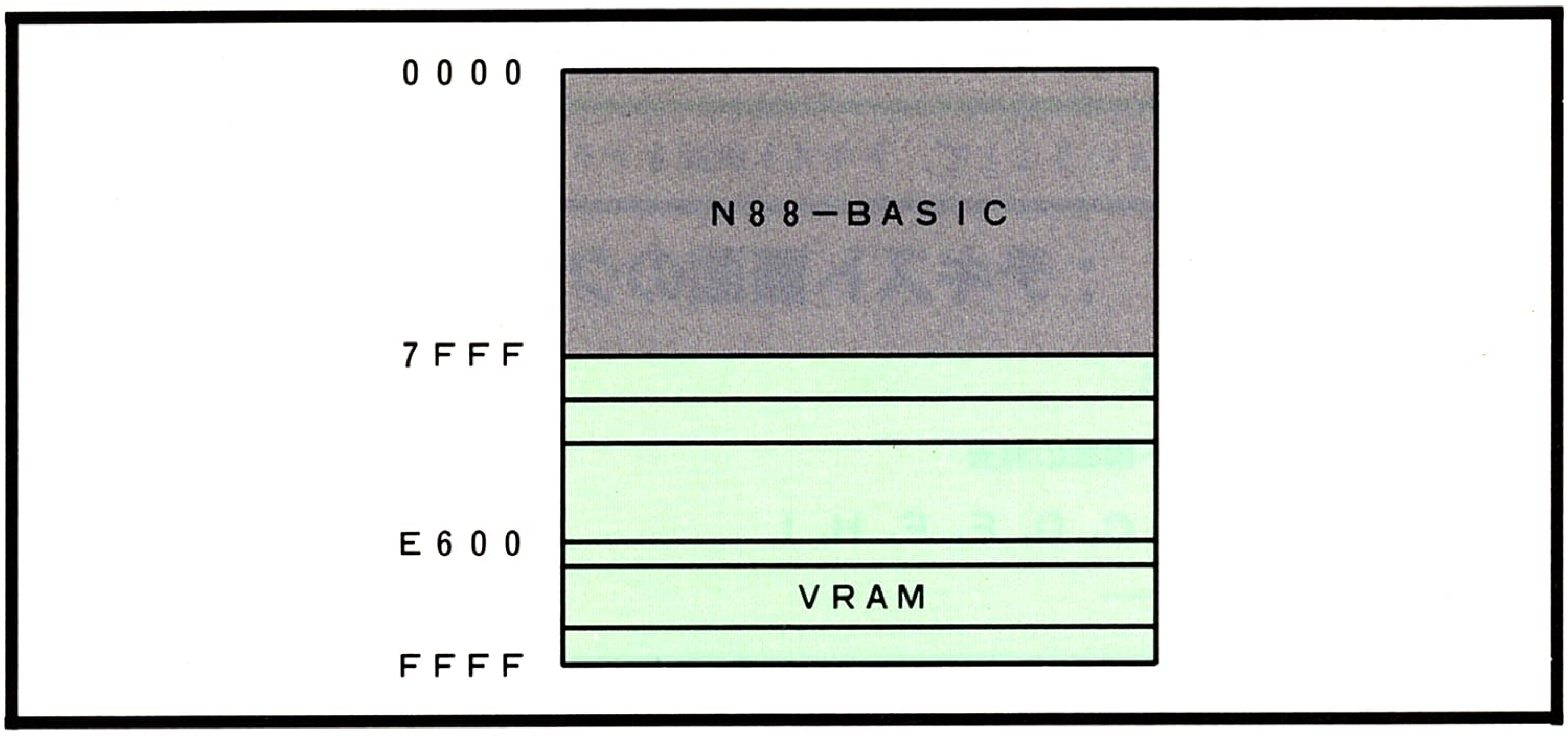
それではなぜ画面をクリアすることができるのでしょうか。
べつに私たちがプログラムを作ったわけではありません。

実は、これは、もともとこういう**「画面をクリアする」**という命令がマイコンの中に書き込まれていたからなのですね。

ＢＡＳＩＣで、**「ＰＲＩＮＴ」**や**「ＣＬＳ」「ＦＯＲ～ＮＥＸＴ」「ＬＩＮＥ」**などが使えるのもまた同じことで、そのような処理をするプログラムがマイコンの中に記憶されているからなのです。

これらはすべて**「ＲＯＭ」**の中に書き込まれているため電源を切ってもプログラムが消えなかったというわけです。

これらは、０００００から７ＦＦＦの中にいろいろつまっています。



００００～７ＦＦＦといえばN88－ＢＡＳＩＣの記憶されている場所でした。

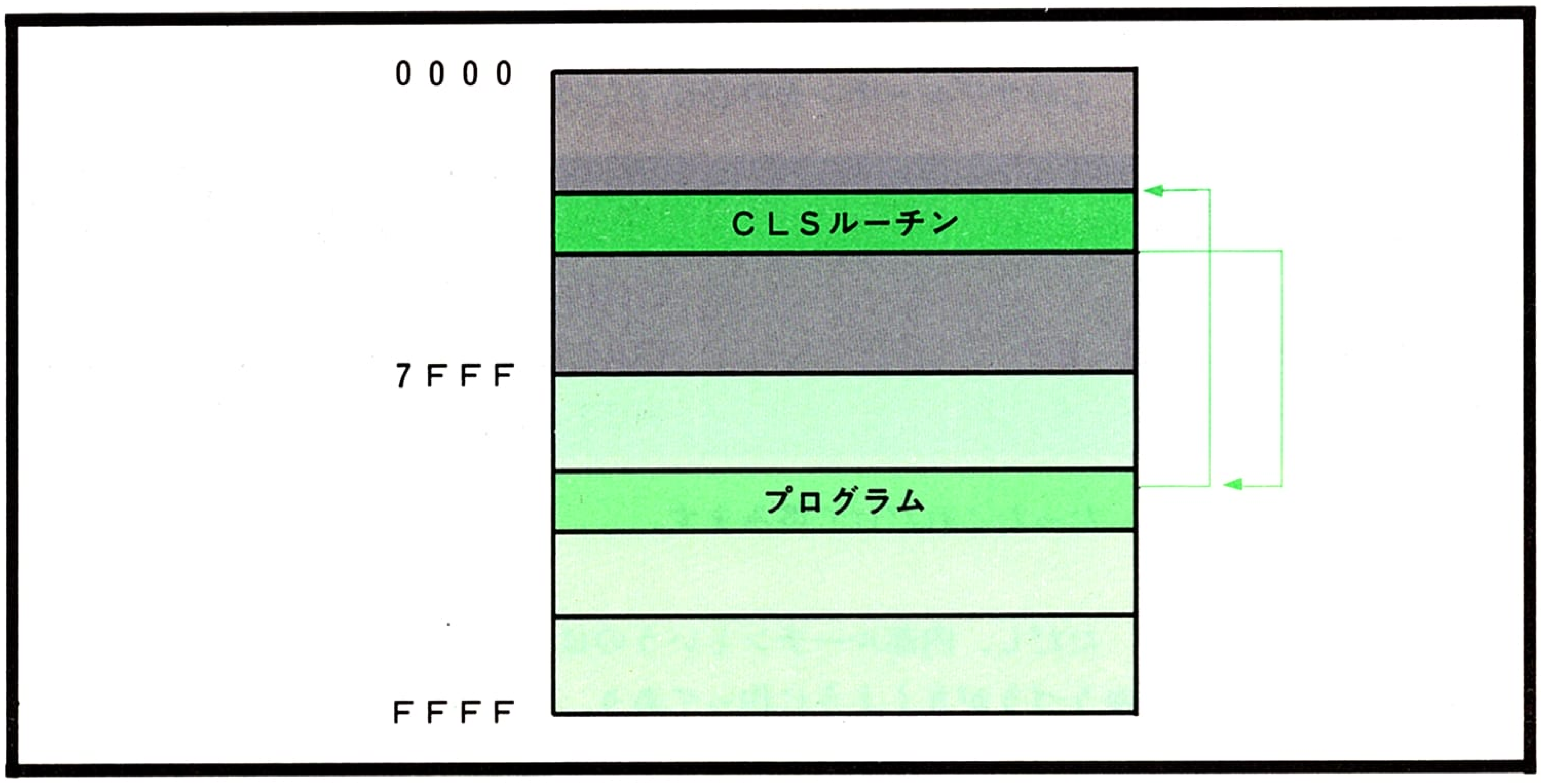
つまり、**ＢＡＳＩＣを走らせている時にマイコンが使うサブルーチンが記憶されているという場所**です。

私たちは、このサブルーチンをちゃっかり借りようというわけなのです。

ＲＯＭに書かれているサブルーチンも、私たちが作るプログラムも同じ「**マシン語**」でできています（当然の話）。

ですから、そのまま「**ＣＡＬＬ**」でサブルーチンを呼ぶことができるのです。

ただし、サブルーチンの中で処理をする時に必要な情報をあらかじめレジスタに入れておかなければいけないこともあります。



3·6　内部ルーチンの例

ということで、テキスト画面をクリアする内部ルーチンです。

5F0EH　：テキスト画面のクリア

アドレス	5F0E
機　能	テキスト画面の消去
レジスタ	A，B，C，D，E，F，H，L
引　数	――――――

アドレス……このサブルーチンの**先頭アドレス**です。
ここから呼び出します。

機　能……このサブルーチンのする処理の内容です。

レジスタ……このサブルーチンを実行した時に使う**レジスタ**です。
実行後、このレジスタの内容は破壊されたりします。

引　数……このサブルーチンを実行する際に必要な**情報**です。
いらない時もあります。

このサブルーチンを使って**画面をクリアするプログラム**を使ってみましょう。

実に簡単。
このサブルーチンを呼び出すだけでいいのですから、

アドレス	マシン語	ラベル	ニーモニック	コメント
			ORG　B000	START
B000	CD　0E　5F		CALL　5F0E	GOSUB ＊CLS
B003	FF		RST　38H	END

たったこれだけで済みます。

ただし、内部ルーチンというのはＢＡＳＩＣで、どんな場合にでもゆうづうがきくように作ってあり、それだけ無駄があります。
したがってスピードが落ちることを覚悟しなければなりません。

次は、ＶＲＡＭアドレスを求めるサブルーチンです。

429DH：キャラクタの座標をVRAMアドレスに変換

アドレス	429D
機　能	キャラクタ座標からＶＲＡＭアドレスを計算しＨＬに入力する
レジスタ	A，F，H，L
引　数	H←横座標（1～80） L←縦座標（1～25）

ＨにＸ座標に１足したもの、ＬにＹ座標に１足したものを入力しておいてからこの内部ルーチンを呼び出すと、**実行後ＨＬレジスタの中にそのＶＲＡＭアドレスの値が入ります。**

たとえば、Ｈに11、Ｌに11を入れてルーチンを呼び出すと、実行後ＨＬレジスタに「Ｆ８８Ｃ」が入力される、というわけです。

アドレス	マシン語	ラベル	ニーモニック	コメント
			ORG B000	START
B000	26 0B		LD H，0B	H←11
B002	2E 0B		LD L，0B	L←11
B004	CD 9D 42		CALL 429D	GOSUB *VRAM
B007	FF		RST 38H	END

```
h]GB000
h]x
A :14 F :P----0-- B :0000 D :EDCC
H :F88C A':14 F':P----0-- B':0000
D':FF7B H':0911 IX:0000 IY:0830
I :F3 PC:B007 SP:E5F9
h]■
```

ただし、これは40字モードの場合の話で、80字モードにするとちゃんとＨＬレジスタに「Ｆ８８２」が入ります。

それは、計算する時に「現在はどちらのモードなのか」ということを判断しているため、そのモードで必要なＶＲＡＭのアドレスが求められるからなのです。

第4章

USR関数

4·1 BASIC vs マシン語

私たちは、こうやっていろいろとマシン語のプログラムを作ってきたわけです。

とは言うものの、これはほんとに初歩の初歩。

本格的に作るとなるとやはりそれなりの努力がいるということを感じさせられます。

マシン語もスピードが速いのはうれしいのですが、作るのにちょっと手間がかかってしまうというのが欠点のようです。

マシン語もＢＡＳＩＣのように作りやすかったらいいと思います。

ところで、こうしてみると両者の長所、短所に気がつくのではないでしょうか。

ＢＡＳＩＣ：作りやすいが、遅い。

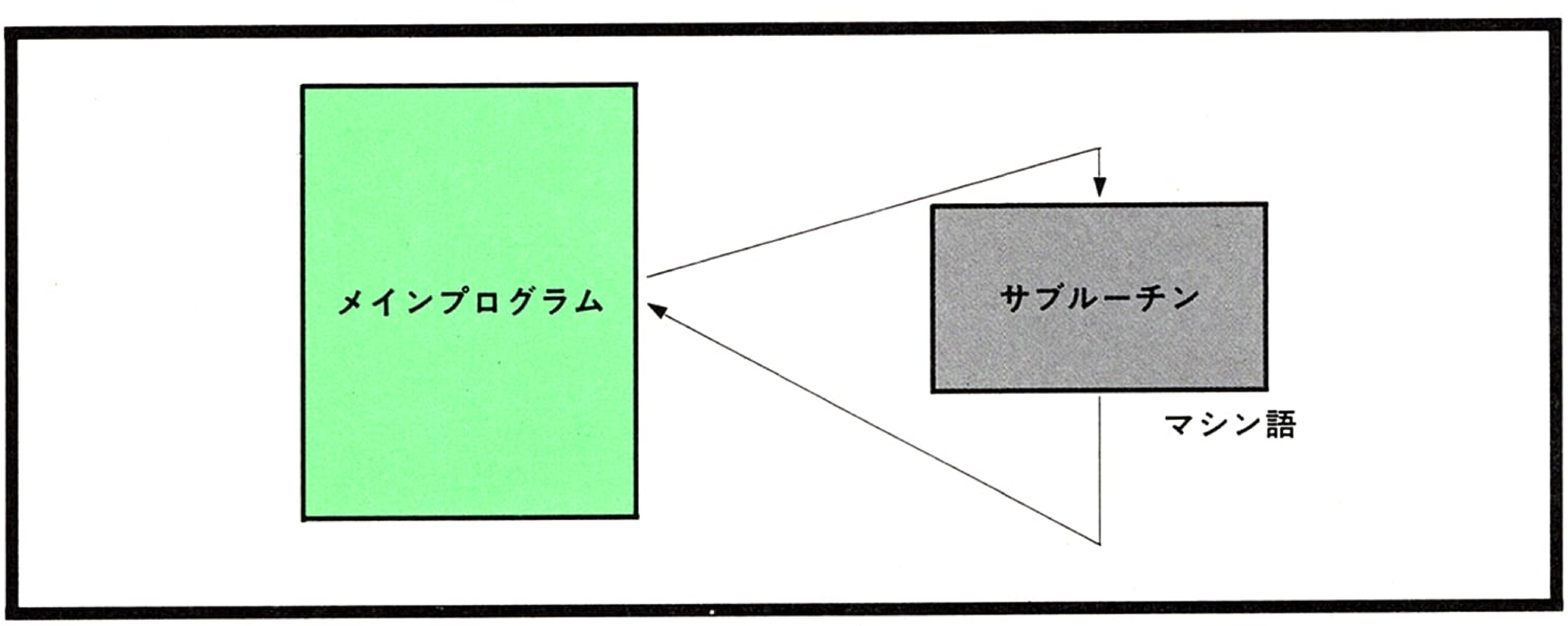
マシン語　：作りにくいが、速い。

そこで、プログラムを作る時それぞれの長所を生かすようにしてやりましょう。

一番重要で複雑な計算が多いメインプログラムを作りやすいＢＡＳＩＣで作ります。

そして、単純な計算でできるわりには時間のかかる絵の表示などの簡単な処理をスピードの速いマシン語で作るわけです。

こうすれば、作りやすくて速いプログラムができるというものです。

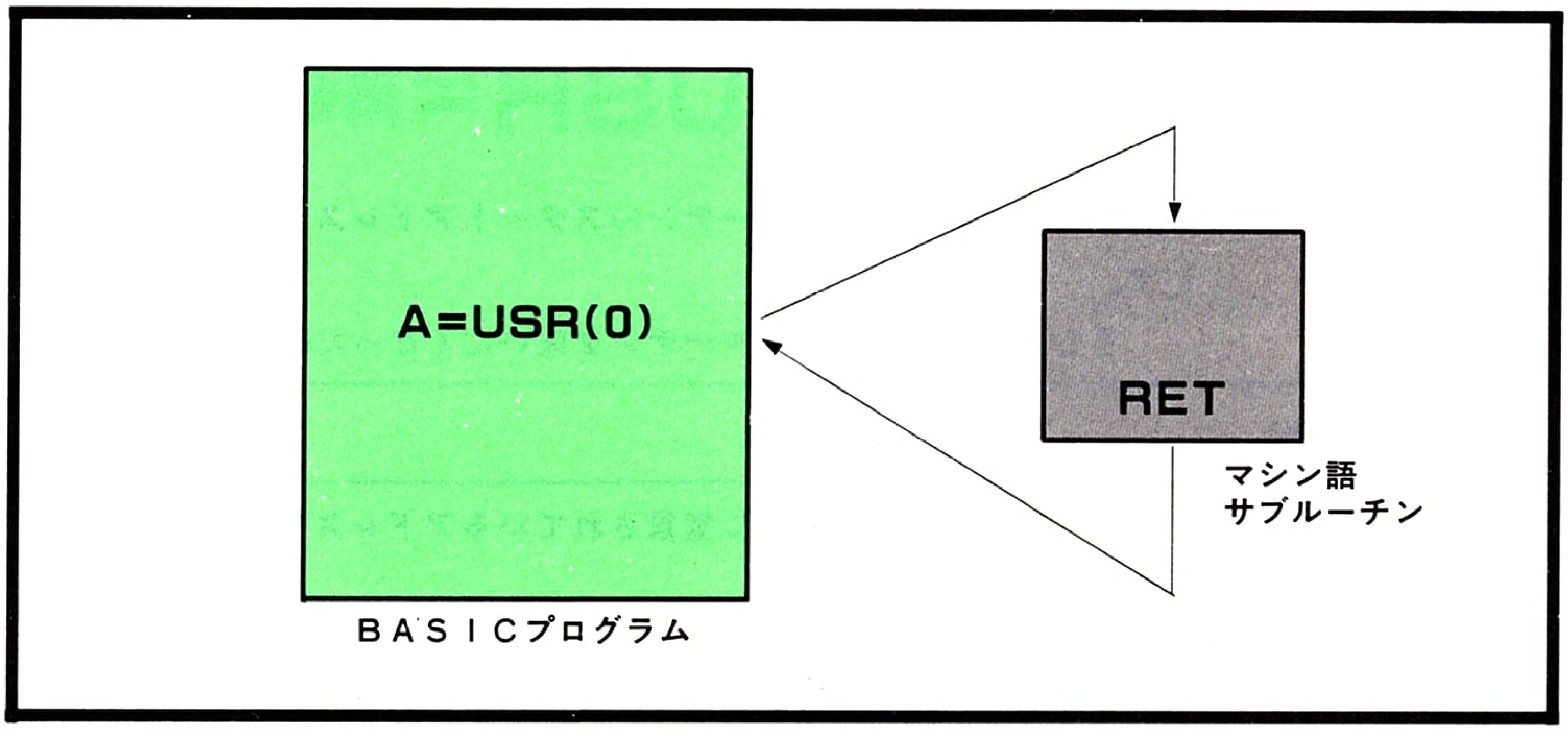


このようなことをするためには、ＢＡＳＩＣとマシン語をつなぐ命令が必要になってきます。
その命令というのが、

USR関数

です。

ＢＡＳＩＣからマシン語のサブルーチンを呼び出すというもので、マシン語の「ＲＥＴ」でＢＡＳＩＣへ戻ります。



つまり、ＵＳＲ関数というのは、

マシン語のＧＯＳＵＢ

というわけです。

4・2　USR関数

ＵＳＲ関数は、

変数＝USR(0)

というようにして使います。

まず、プログラムの最初に、

DEF USR＝開始アドレス

として、マシン語サブルーチンの**スタートアドレス**を宣言しておきます。

そして、マシン語サブルーチンを使いたくなった時、

```
A=USR(0)
```

として呼び出せば、**すでに宣言されているアドレスへとＪＵＭＰ**するわけです。

たとえば、画面をクリアするサブルーチンを呼び出してみましょう。

「ＤＥＦ　ＵＳＲ」でスタートアドレスを宣言してから呼び出します。

この時、アドレスは16進数ですので、**数字の前に〝＆Ｈ〟をつける**ことを忘れないで下さい。

```
10 DEF USR=&H5F0E
20 A=USR(0)
30 END
```

ＲＵＮすると、画面はクリアされます。

このようにしてＢＡＳＩＣのプログラムからマシン語のサブルーチンを楽々と呼び出すことができるわけです。

10個のUSR関数

ですが、宣言してから使うのでは「GOSUB」のように自由に好きな所を呼び出すことができません。
できないこともないのですが、そのたびに宣言しなおさなくてはなりません。

そこで「USR関数」には０〜９までの10個のUSR関数が用意されていて、**それぞれに別々の違ったサブルーチンを宣言できるようになっている**のです。

つまり、10個までなら、宣言しなおさなくても好きなところを自由に呼び出せるというわけです。

DEF USR 番号=開始アドレス

と宣言して、

A=USR 番号 (0)

と番号で呼び出します。

先ほどのように番号をつけない場合は、０番のUSR関数と解釈されます。

★focus★

サブルーチン、サブルーチンと言っていますが、実際は「命令」とか、いわゆる「関数」と考える方がいいでしょう。
呼び出すからと言っても、そのルーチンの中で使われている変数がメインプログラムの中で使用されているか、などと気にする必要がないからです。
画面を消したいなら「画面を消す」、音を出すなら「音を出す」というルーチンのスタートアドレスさえわかれば、あとはそのルーチンの中が一体どうなっているのかなどと考えなくても良いわけです。

要は、そのルーチンにできる仕事の内容が重要なのですから……。

4・3　データの連絡

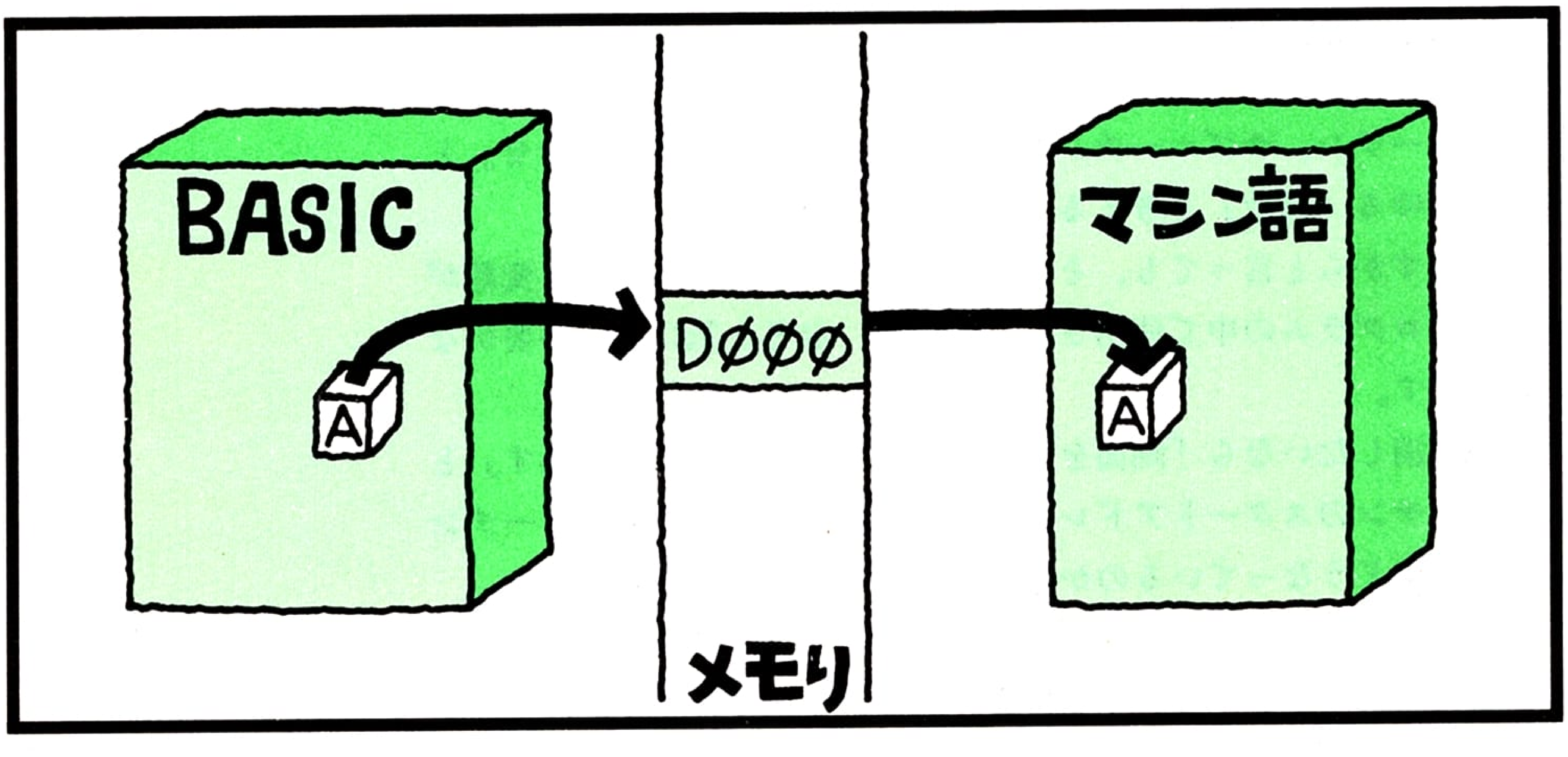
「画面のクリア」とか「ＢＥＥＰ音を出す」などは外部からの情報がなくても実行できます。
する仕事が決まっているからです。

でもほとんどの処理にはデータが必要です。仕事の情報がはっきりしていないからです。
たとえば、ＢＡＳＩＣで、
「ＰＲＩＮＴ」の場合、表示する内容が必要です。
「ＬＯＣＡＴＥ」なら、Ｘ座標、Ｙ座標のデータが必要ですし、
「ＣＯＬＯＲ」にしても「ＬＩＮＥ」にしても同じことです。

マシン語で作ったプログラムをＢＡＳＩＣから呼び出す時も同じことで、やはりデータの受けわたしが必要になることもあります。
そこで、**「データの連絡方法」**について話すことにしましょう。

POKE & PEEK

ＢＡＳＩＣでもマシン語でも使用しているもの、それは**「メモリ」**です。
どちらもこの「メモリ」に収納されている数値を読み出したり、また逆に書き込んだりする命令が存在します。
そこで「ＢＡＳＩＣの変数」から「マシン語のレジスタ」へデータを送る時、**「メモリ」を通して行えばいい**というわけです。



たとえば、**「変数Ａの中身をレジスタＡ」**に入力してみます。

まず、ＢＡＳＩＣで

```
POKE &HD000,A
```

と、たとえば「Ｄ０００番地」のメモリにデータを入れておきます。

そうしておけば、今度マシン語のプログラムに移っても、

```
LD  A,(D000)
```

とすることによってＡレジスタに代入することができるのです。

反対に、

```
LD  (D000),A
```

としてＡレジスタの値をメモリに一時記憶させ、

```
A=PEEK(&HD000)
```

として変数Ａにレジスタの中身を代入することもできるのです。

★ for example ★

それでは、**マシン語で「ＬＯＣＡＴＥ　１０，１０」の位置にハートを表示**させてみましょう。

Ｘ＝10，Ｙ＝10のデータはＢＡＳＩＣから送るものとします。

まず最初にマシン語の部分から作ってみます。

最初はＢＡＳＩＣから送られて来たＸ座標Ｙ座標の値を読み込みます。データが保管される場所を、今回は「Ｃ０００」「Ｃ００１」にします。

ほんとはどこでもいいのですが、マシン語やＢＡＳＩＣのプログラムが書かれているメモリを使うと暴走するおそれがあるので、あきのメモリを使うわけです。

ＬＤ命令を使ってＨにＸ座標、ＬにＹ座標を代入します。

アドレス	マシン語	ラベル	ニーモニック	コメント
			ORG B000	START
B000	3A 00 C0		LD A, (C000)	H←(C000)
B003	67		LD H, A	
B004	3A 01 C0		LD A, (C001)	L←(C001)
B007	6F		LD L, A	

あとの処理は、内部ルーチンを適当に使って、

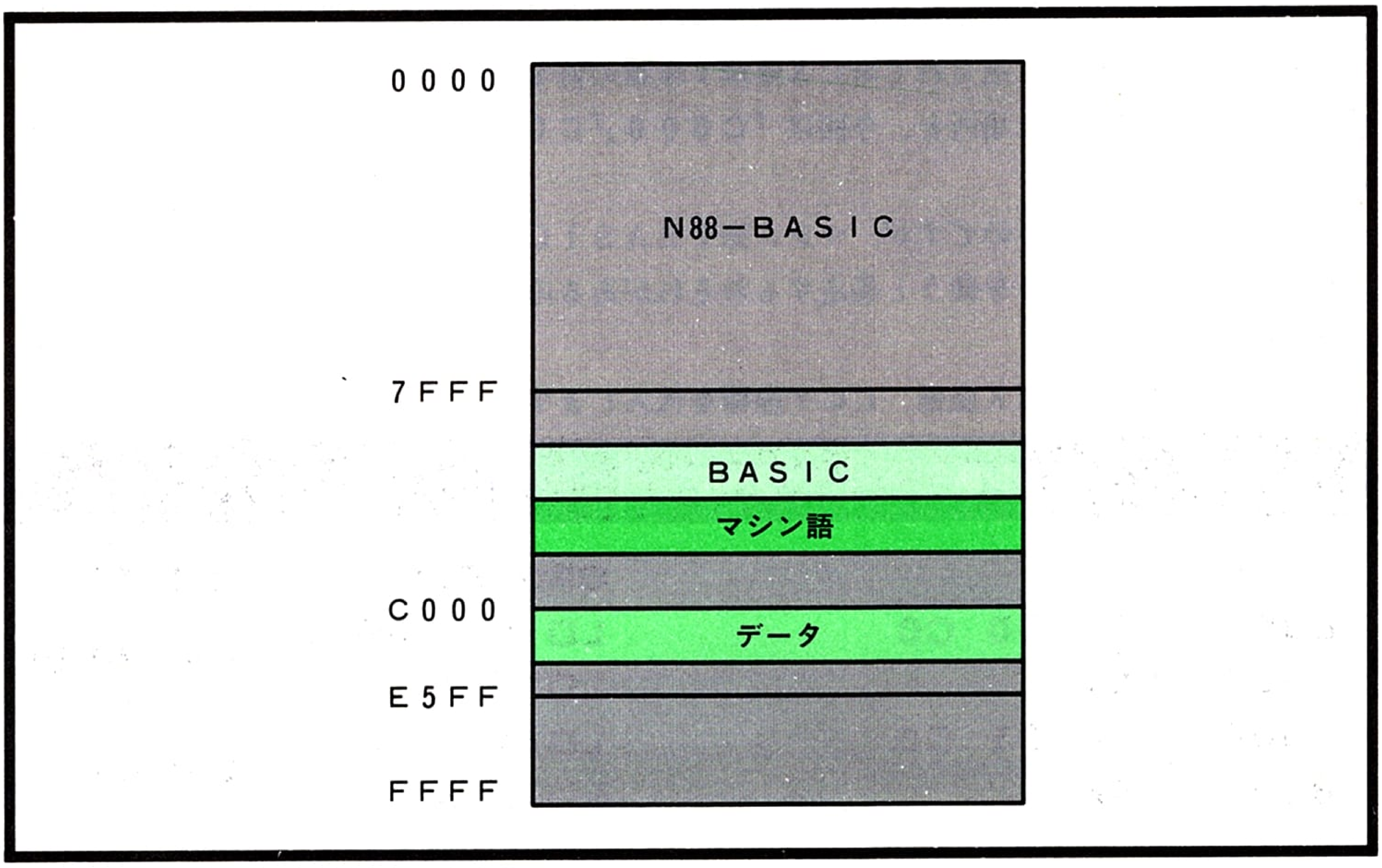
アドレス	マシン語	ラベル	ニーモニック	コメント
B008	24		INC H	H←H+1
B009	2C		INC L	L←L+1
B00A	CD 9D 42		CALL 429D	VRAMアドレス計算
B00D	3E E9		LD A, E9	A← "♥"
B00F	77		LD (HL), A	(HL)← "♥"
B010	C9		RET	BASICへ戻る

としてできあがりです。

ＢＡＳＩＣで、

```
10 CLEAR ,&HAFFF
20 DEF USR=&HB000
30 X=10 :Y=10
40 POKE &HC000,X
50 POKE &HC001,Y
60 A=USR(0)
70 END
```

を実行させればちゃんと走ります。



A=USR(0)

もう１つの方法に「A＝USR(０)」をそのまま使うという手があります。

基本的には「ＰＯＫＥ」や「ＰＥＥＫ」を使った時とほとんど同じです。
つまり、**「メモリ」を通してデータをやりとりする**のです。

これには、

整数型
単精度数値型
倍精度数値型
文字型

と、４種類ぐらいあります。

しかし、使われるのはほとんど整数型ですし、また最もわかりやすいのも整数型なので**「整数型」**についてだけ話します。

変数=USR（整数型数値）

ＢＡＳＩＣからマシン語を呼ぶ時、

```
A=USR(0)
```

という命令を使っていました。
この丸カッコの中にデータを入れてマシン語に送ってやろうというわけなのです。
整数型というのは、この丸カッコの中を整数型の数値か、または「ＤＥＦＩＮＴ」等で整数型変数に宣言された変数を使用するということです。
たとえば「1000」という数字をＢＡＳＩＣからマシン語に送りたいとします。
そんな時は、

```
A=USR(1000)
```

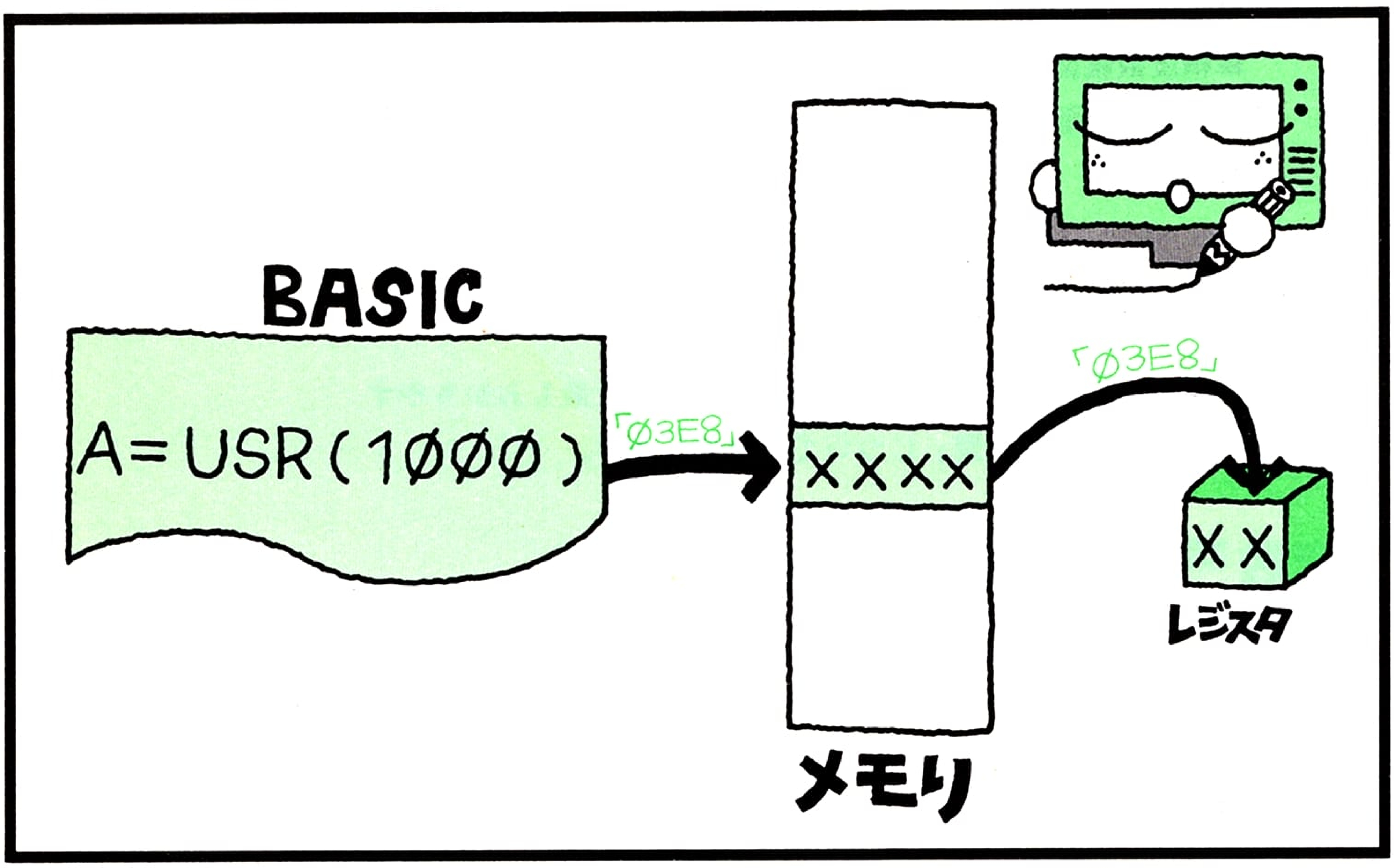
また、

```
100 DEFINT X
110 X=1000
120 A=USR(X)
```

とするだけで送れるのです。

「1000」という10進数が16進数の「０３Ｅ８」という形でメモリに書き込まれます。

そしてマシン語になった時メモリから「０３Ｅ８」という形で読みとればいいのです。



〈注〉ＵＳＲ関数でパラメータをマシン語に引き渡すと、整数型の場合その値は「浮動小数点アキュムレータ」（ＦＡＣ）の５バイト、６バイト目に格納され、この格納番地はＨＬレジスタにしまわれています。つまりＨＬレジスタの指し示す番地を見ればパラメータを得ることができるのです。

ここで問題になるのが、**メモリの「どこ」に書き込まれるか**です。

これは、私たちが勝手に決めることはできません。

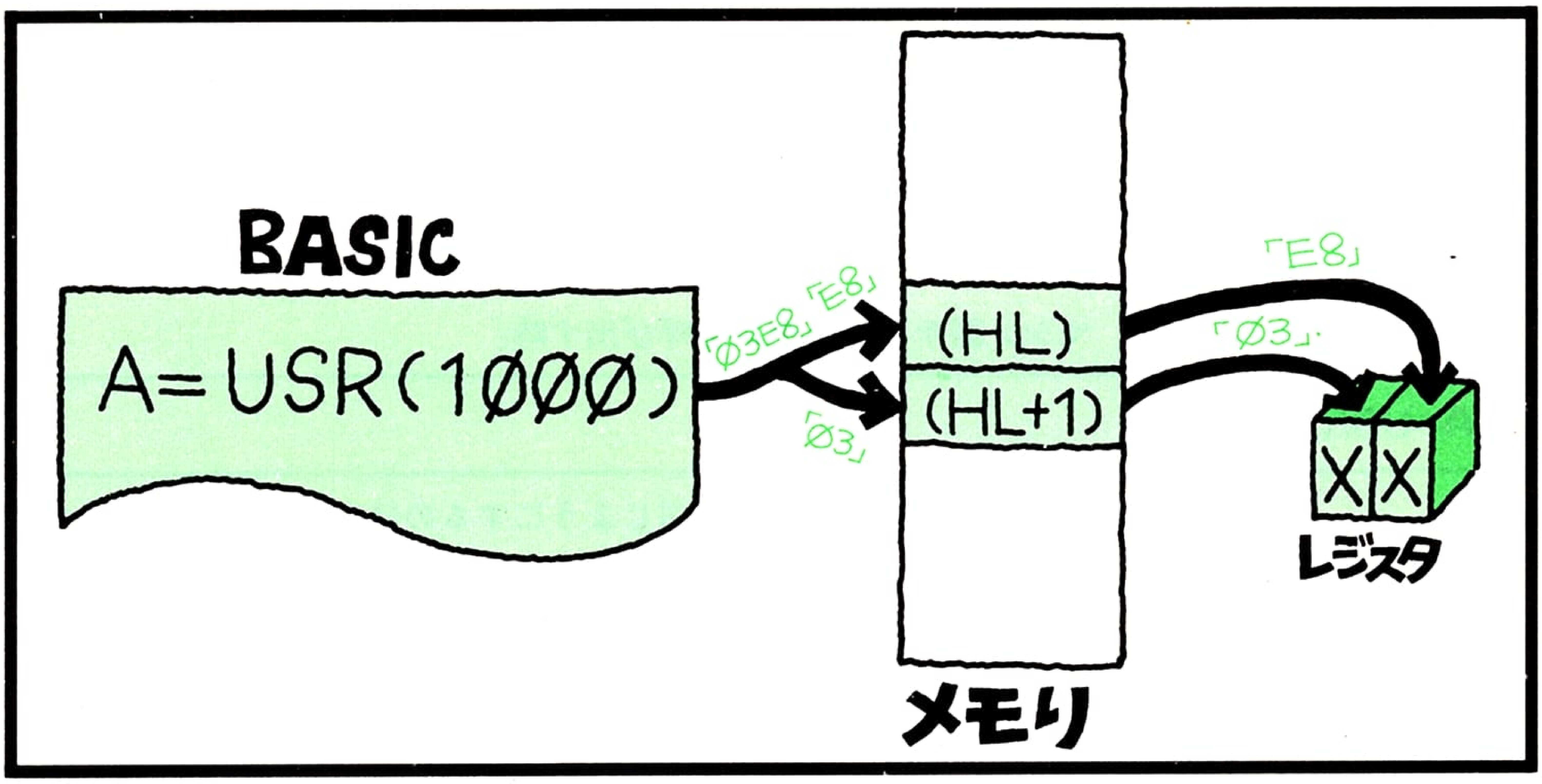
「ＰＵＳＨ」や「ＰＯＰ」の時と同じようにＣＰＵが自分の都合のいい場所に書き込むのです。

その後、書き込んだメモリの番地をＨＬレジスタペアに知らせてくれます。

私たちは、**ＨＬレジスタペアが示す番地のデータを見れば良い**というわけです。

ここで扱う数値は４ケタの16進数ですから、**データはＨＬレジスタペアが指している番地とその次の番地に書き込まれています。**

また、上位２ケタ下位２ケタが逆になって書き込まれていることも忘れてはなりません。



まとめ

ＤＥＦ　ＵＳＲ〔番号〕＝〈開始アドレス〉

- マシン語サブルーチンのスタートアドレスを宣言する。

Ａ＝ＵＳＲ〔番号〕（０）

- マシン語サブルーチンに飛ぶ。

Ａ＝ＵＳＲ〔番号〕（〈整数型〉）

- ＢＡＳＩＣからマシン語サブルーチンに飛ぶ時、同時にデータも送る。
- データは、ＨＬレジスタが指すアドレスとその次のアドレスの２バイトに格納される。
- ＨＬレジスタが指すメモリには下位２ケタ、その次のアドレスには上位２ケタが格納される。

ＲＥＴ

- ＢＡＳＩＣからマシン語のサブルーチンを実行していた時、ＢＡＳＩＣへ戻る。

ＢＡＳＩＣへ

反対に、マシン語からＢＡＳＩＣへ戻る時のデータの受けわたしについて考えてみます。

これも「ＰＯＫＥ」や「ＰＥＥＫ」を使った時と原理は同じです。

メモリに書き込まれたデータをＢＡＳＩＣの変数に読み出すわけです。

マシン語サブルーチンを呼び出す時、

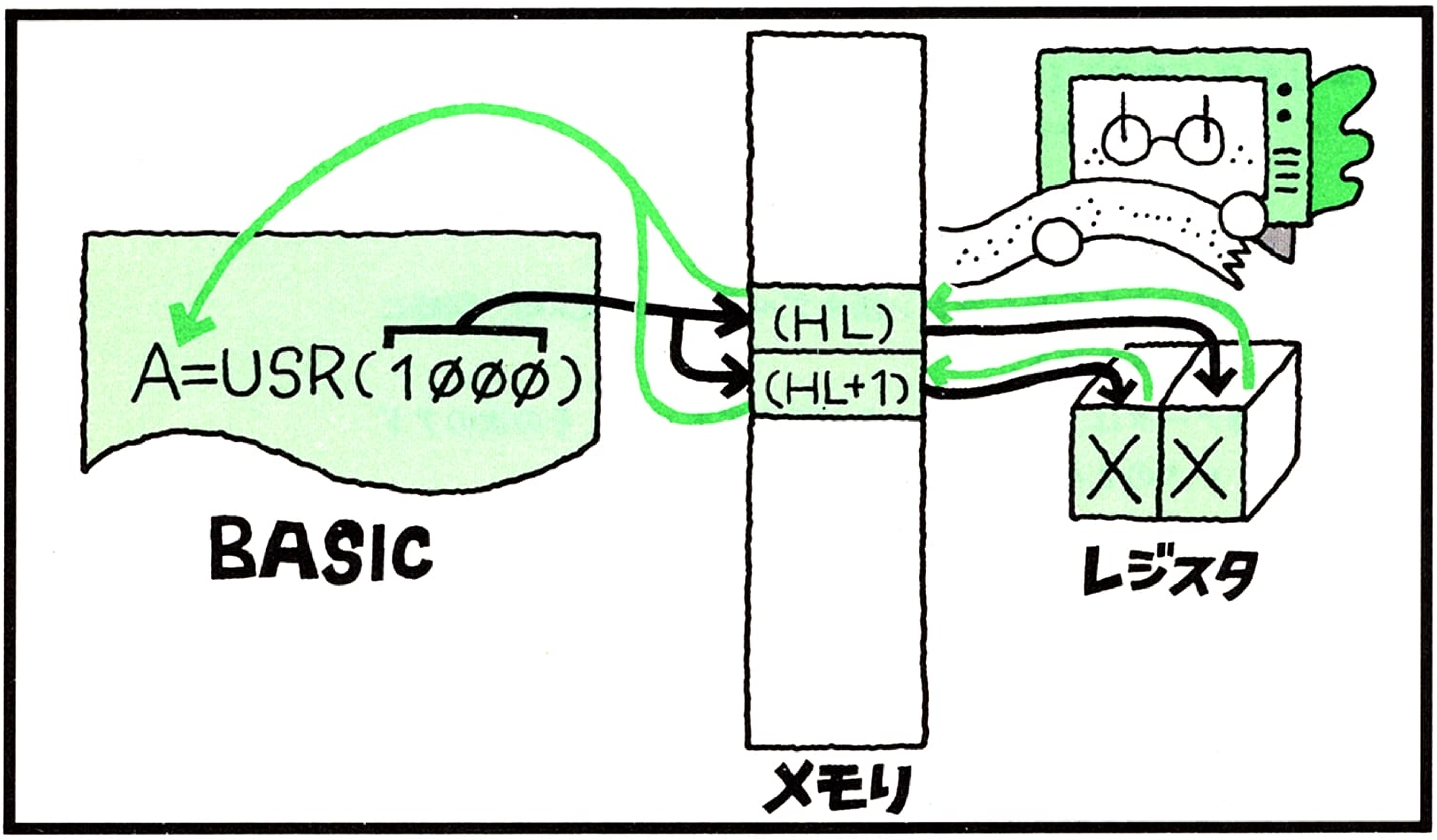
```
A=USR(0)
```

と、まるで「ＬＥＴ文」と同じようにするのはそのためだったのです。

つまり、**マシン語から送られて来たデータを変数に**（この場合ならＡに）**代入**しようというのです。

この時ＢＡＳＩＣの変数に入るデータの番地は、マシン語サブルーチンを呼んだ時のＨＬレジスタに書かれてあったものです。

ＢＡＳＩＣからマシン語への連絡も、マシン語からＢＡＳＩＣへの連絡も同じ場所で行われているということなのです。



★ for example ★

練習のために、「＋１」**をするマシン語サブルーチン**を作ってみます。

まず、ＢＡＳＩＣからデータを読み込んで、

アドレス	マシン語	ラベル	ニーモニック	コメント
			ORG B000	START
B000	E5		PUSH HL	HLを保管
B001	5E		LD E, (HL)	E←下2ケタ
B002	23		INC HL	
B003	56		LD D, (HL)	D←上2ケタ

「＋１」を計算します。

アドレス	マシン語	ラベル	ニーモニック	コメント
B004	13		INC DE	DE←DE+1

その後、ＢＡＳＩＣへ戻る前にデータを書き込めば、

アドレス	マシン語	ラベル	ニーモニック	コメント
B005	E1		POP HL	HLを返却
B006	73		LD (HL), E	(HL)←E
B007	23		INC HL	A←
B008	72		LD (HL), D	(HL+1)←D
B009	C9		RET	BASICに戻る

これで、できあがりです。

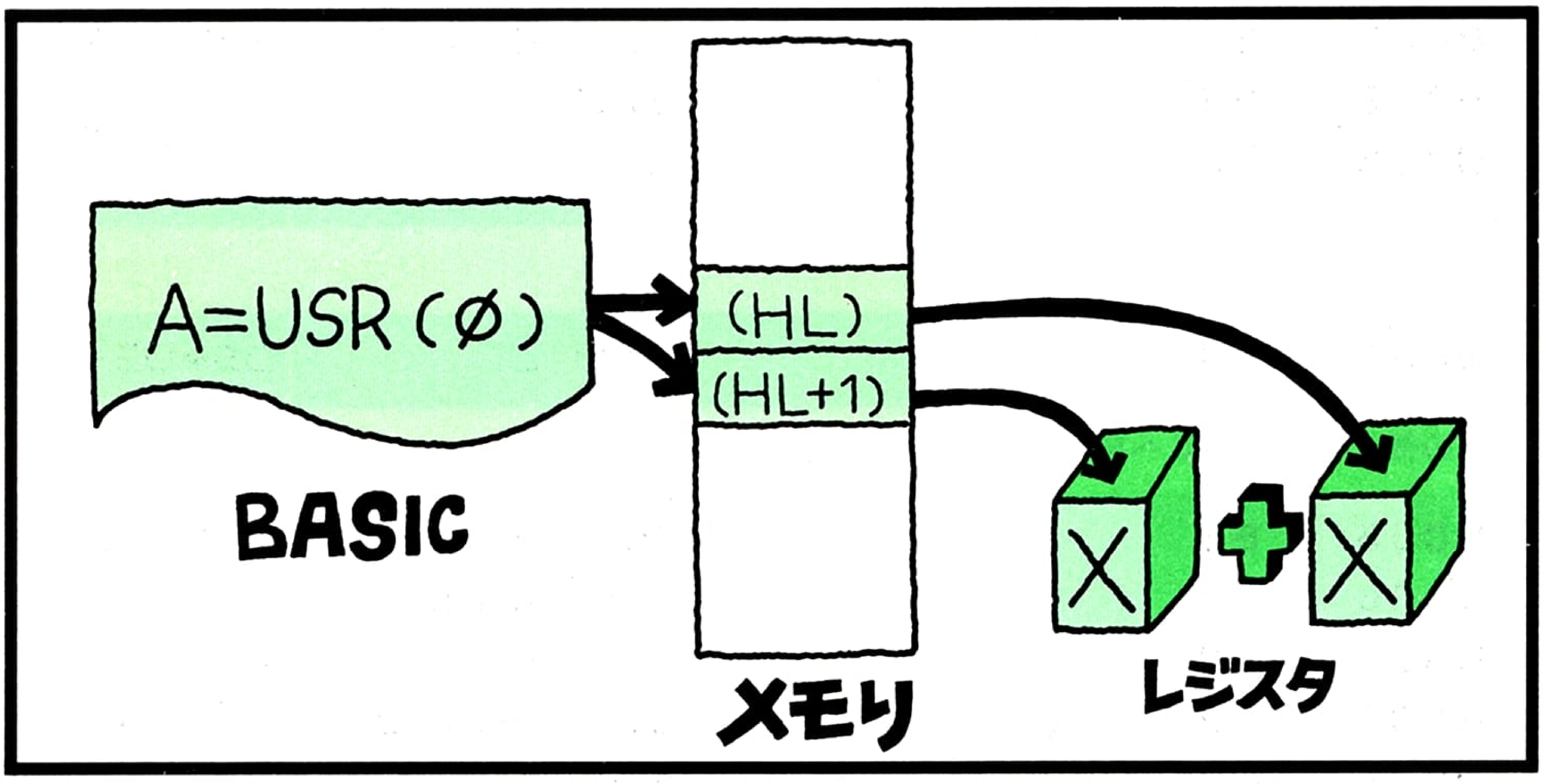
```
10 CLEAR ,&HAFFF
20 DEF USR=&HB000
30 DEFINT A-Z
40 X=1000
50 A=USR(X)
60 PRINT X,A
Ok
run
 1000          1001
Ok
█
```

4・4　X, Yデータの連絡

ところが先ほどの方法ではデータをたった１つしか受けわたしできません。

「ＬＯＣＡＴＥ　Ｘ，Ｙ」のようにデータを２つ以上使う場合は使えないのでしょうか？

そこで１つ便利な方法を紹介しておきましょう。



「ＨＬ」と「ＨＬ＋１」が指す２バイトのメモリを使ってデータの受けわたしを行っていました。

それは、１バイトだけでは０～255までしか表わせないので、まともな計算ができるように２バイトをつなげて－32768～＋32767までの数を表わそうとしたためです。

しかし、「ＬＯＣＡＴＥ」のような場合０～255の数で十分足ります。なにもわざわざ２バイトの数で表す必要はありません。

そこで、「ＨＬ」と「ＨＬ＋１」が指すメモリを別々の違ったデータだと考えるのです。

そうすれば、２コのデータを扱えることになるからです。

データ１コ　（－32768～＋32767）
データ２コ　（０～255），（０～255）

もっとデータを必要とする場合は「ＰＯＫＥ」などを使ってデータを受けわたす方法が考えられます。

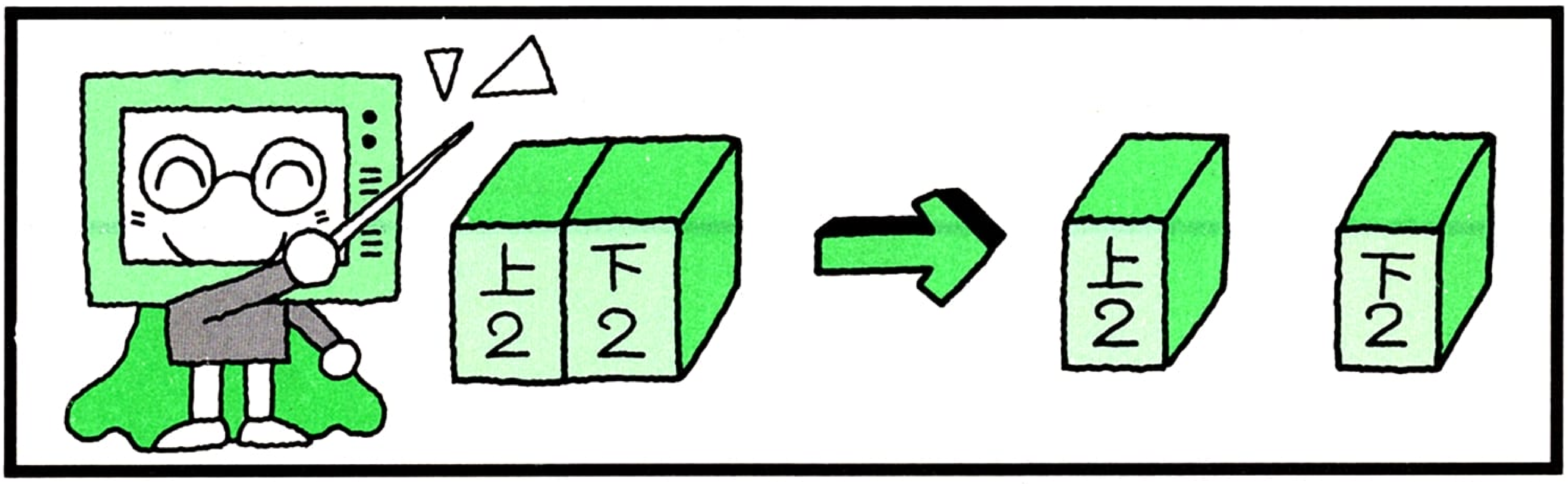
&H100倍

2コのデータを受け渡すといっても、

```
A=USR(0)
```

この方法を使うのですから、1つのデータにまとめる必要があります。

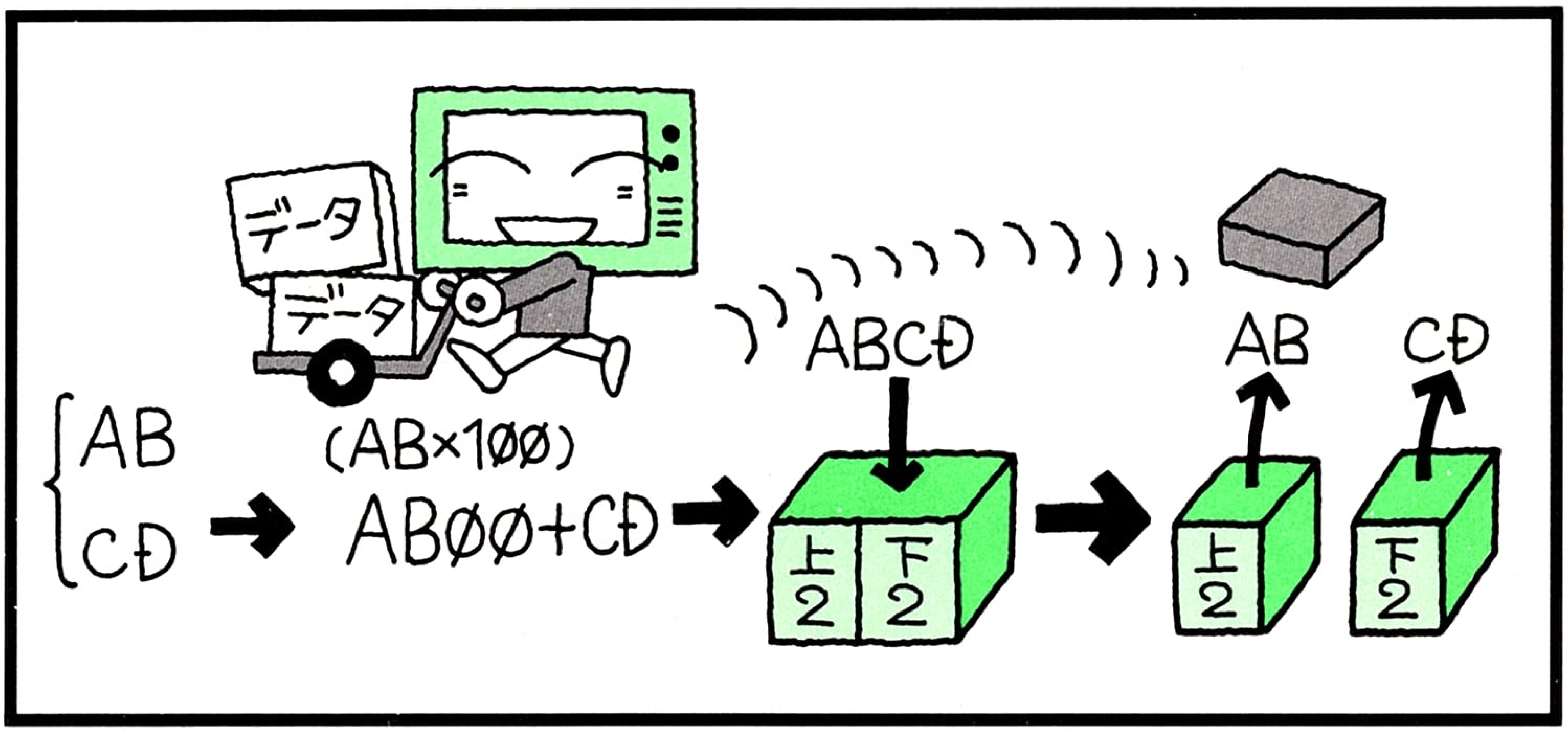
16進数で考えた場合、上2ケタと下2ケタが別々のデータになるわけです。したがって、上2ケタに1つのデータ、下2ケタにもう1つのデータが現われるようにすればいいのです。



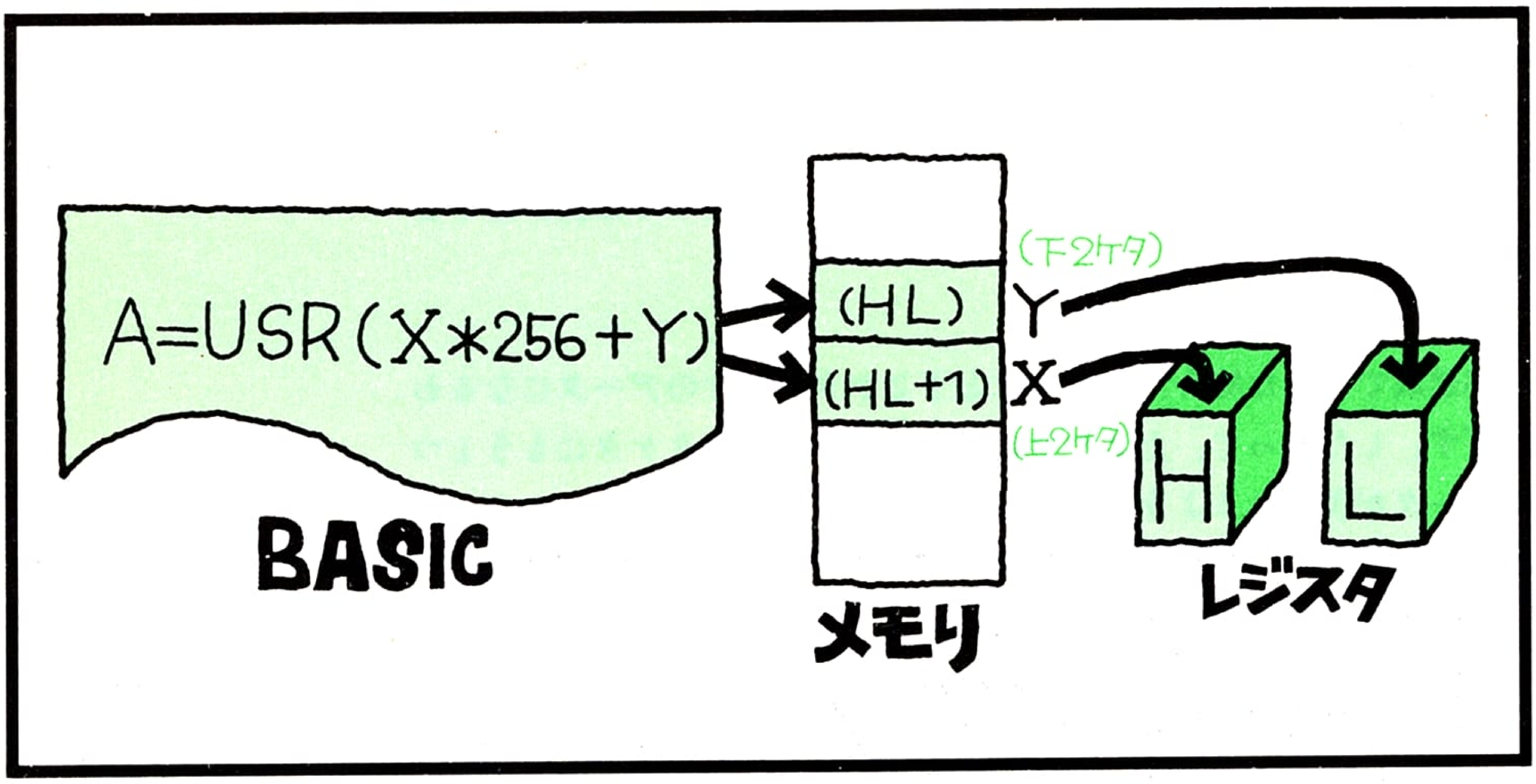
10進数で、「1」を2ケタくり上げるのに「100倍」すれば良いのと同じように、16進数でも2ケタくり上げるのに「&H100倍」します。

つまり、&H100倍（10進数で256倍）して2ケタくり上げた数値が上2ケタになるわけです。

これにもう1つのデータを足せばそれが下2ケタになります。



こうしておけばメモリから2つにわかれたデータを取り出すことができるのです。

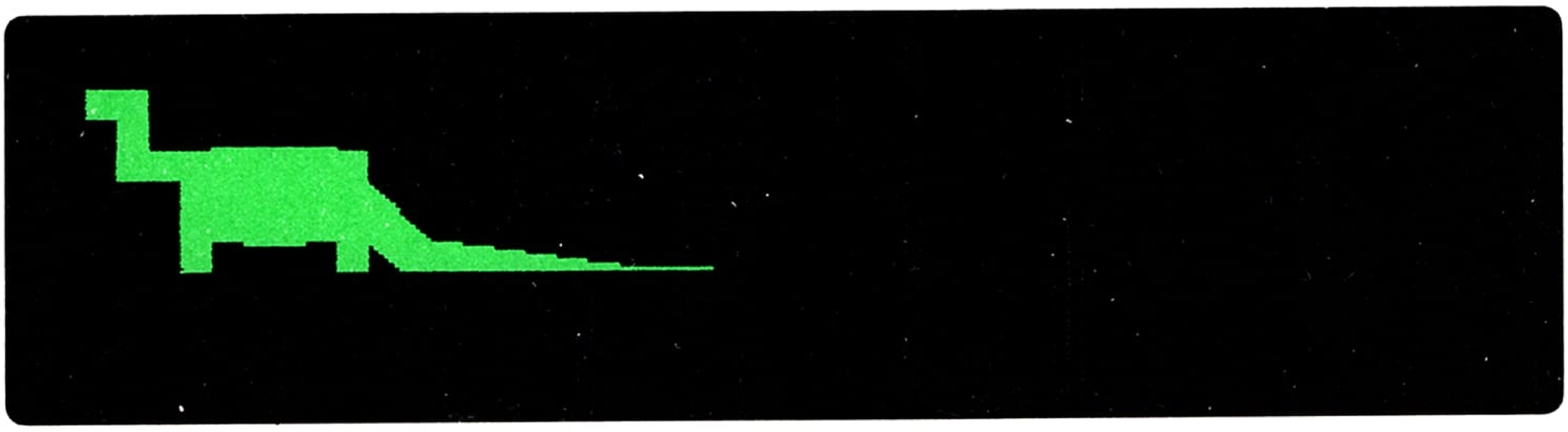


第5章

恐竜のいる街

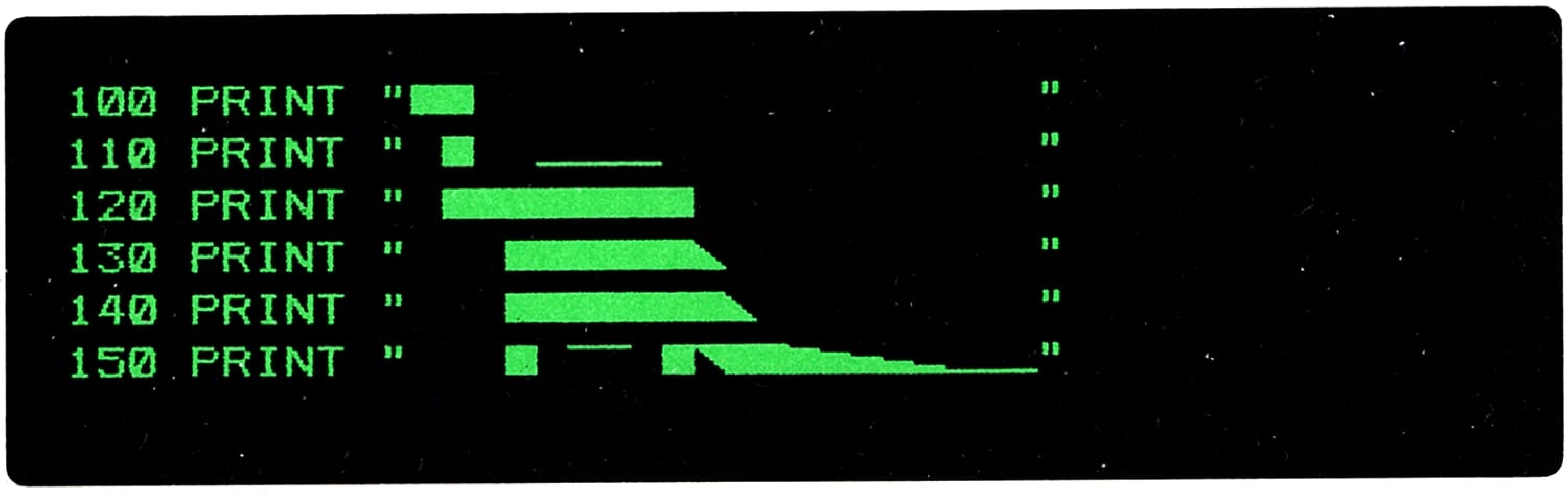
5・1　ブロントザウルス

それでは具体的にプログラムを作っていくことにしましょう。



こんな「ブロントザウルス」を表示させるプログラムを作ってみます。

ＢＡＳＩＣならすぐに、



という方法を思いつきます。

ところが相手はマシン語です。こう簡単には行きません。
もうちょっとくわしくしてやらなくてはいけないのです。

最初大まかに考え、次第に細かい所まで考えて行くことにします。

ブロントザウルスを表示させる

これが、これから作ろうとしているプログラムです。

表示させる時は、**「どこに」**も必要ですからこう決めておきます。

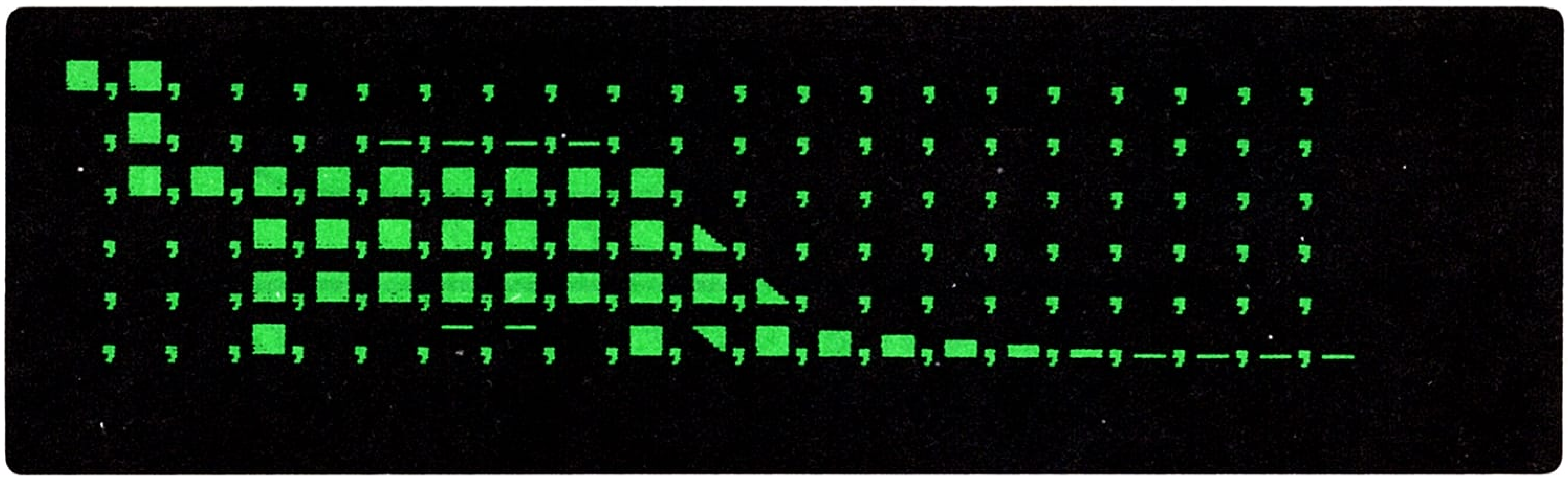
画面の左上に、ブロントザウルスを表示させる

データ作成

表示させるといってもいっぺんには表示できません。ＬＤ命令を使って行うので１つずつしか表示できないのです。

そこで、表示させる絵を縦６×横20の長方形に並べたタイルでできていると考えてデータを作ってみます。

データを



と考えるのです。

これを**アスキーコード**になおして、

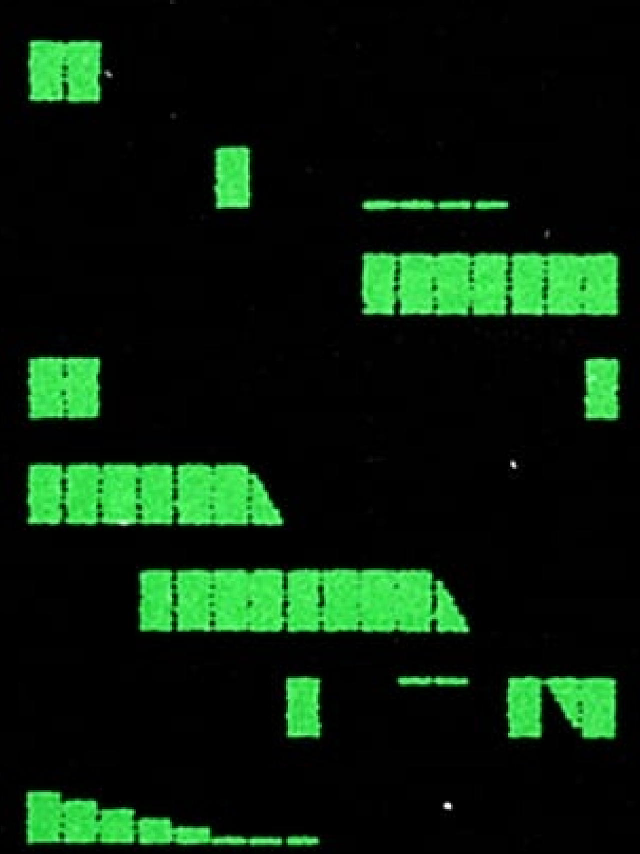
```
87,87,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
00,87,00,00,00,80,80,80,80,00,00,00,00,00,00,00,00,00,00,00
00,87,87,87,87,87,87,87,87,87,00,00,00,00,00,00,00,00,00,00
00,00,00,87,87,87,87,87,87,87,E5,00,00,00,00,00,00,00,00,00
00,00,00,87,87,87,87,87,87,87,87,E5,00,00,00,00,00,00,00,00
00,00,00,87,00,00,94,94,00,87,E6,87,86,85,84,83,82,81,80,80,80
```

とします。

それではメモリ上にデータを書き込んでみましょう。

たとえば、Ｃ０００から書き込んでゆくことにします。

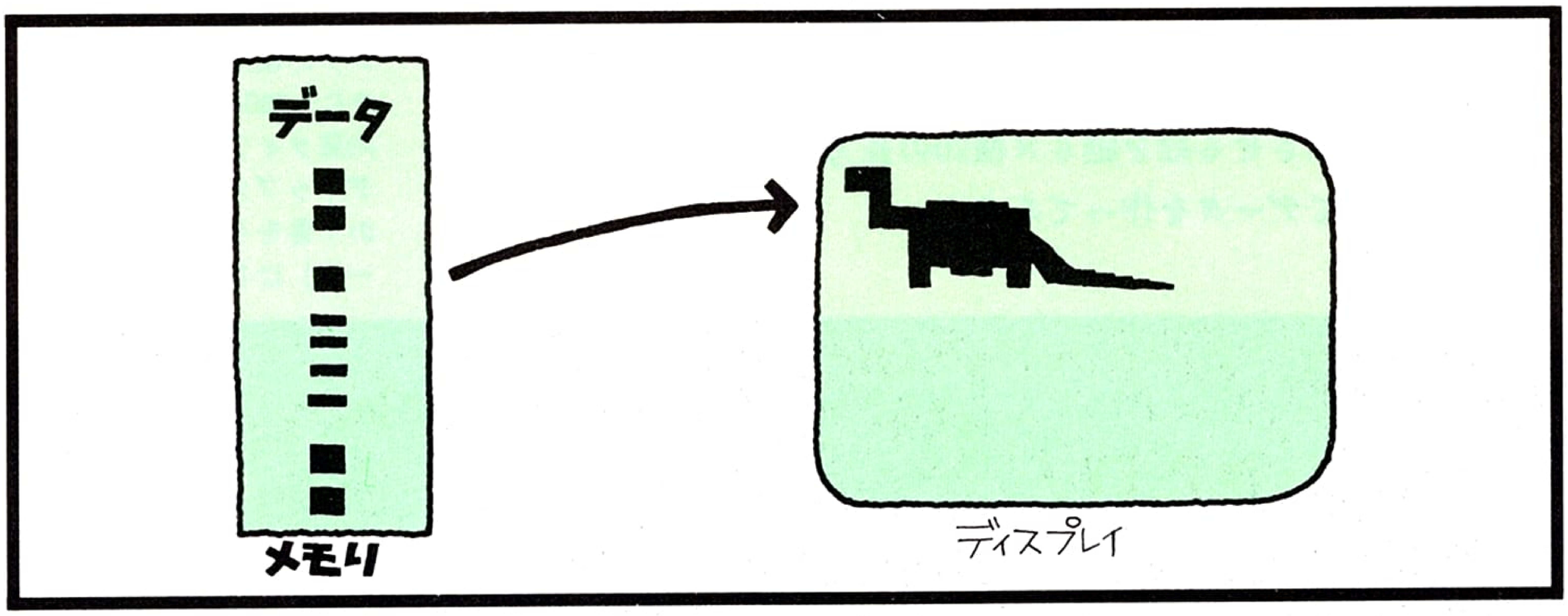
```
C000 87 87 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C010 00 00 00 00 00 87 00 00 00 80 80 80 80 00 00 00
C020 00 00 00 00 00 00 00 00 00 87 87 87 87 87 87 87
C030 87 87 00 00 00 00 00 00 00 00 00 00 00 00 00 87
C040 87 87 87 87 87 87 E5 00 00 00 00 00 00 00 00 00
C050 00 00 00 87 87 87 87 87 87 87 87 E5 00 00 00 00
C060 00 00 00 00 00 00 00 87 00 00 94 94 00 87 E6 87
C070 86 85 84 83 82 81 80 80
```



〈注〉本章ではマシン語プログラムをＢ０００番地以降に書き込みますが、このエリアはＮ88ＤＩＳＫ－ＢＡＳＩＣと重複するためディスクの使用はできません。ＰＣ－8801mkIIのディスク内蔵タイプをご使用の方は、ディップスイッチＳＷ－２の７番をＯＦＦ（ＲＯＭベース）にしてください。

イメージを浮かべる

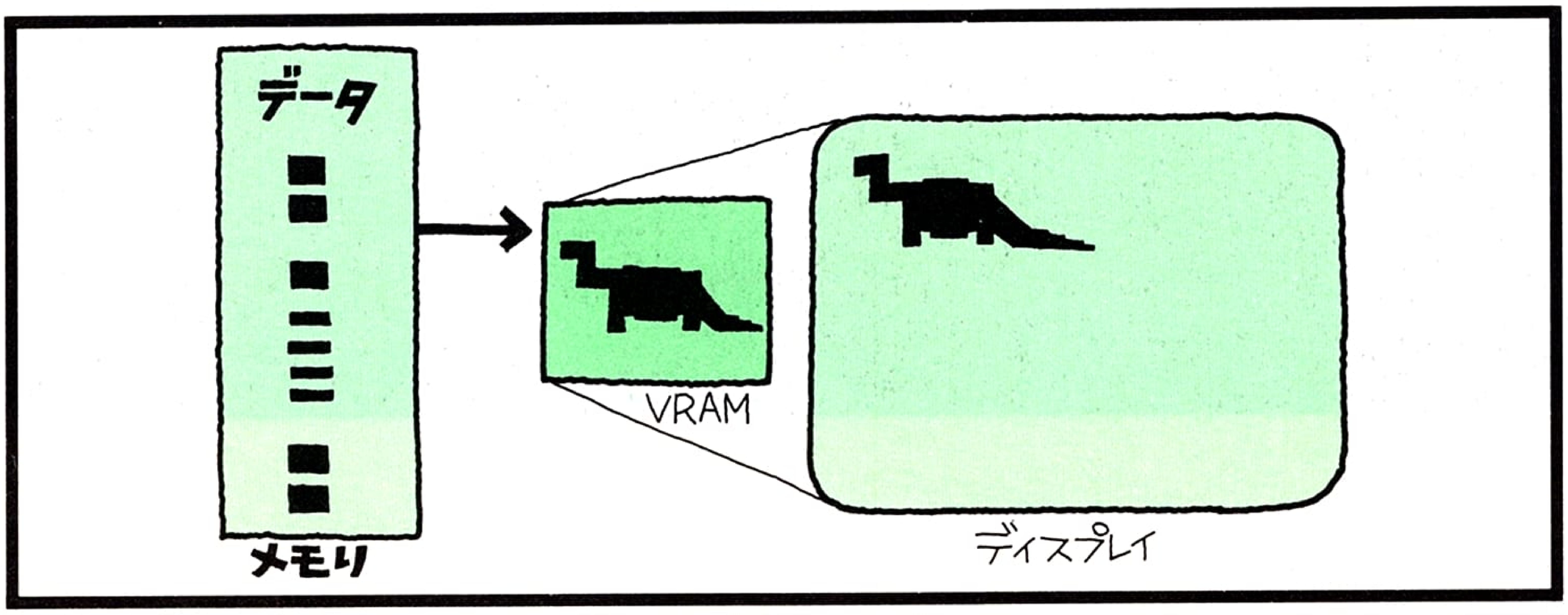
これからどんなことをしたいのかだいたい想像してみます。



メモリに順番に並べられたデータを画面に表示させるのです。

「画面に表示する」というのは**「ＶＲＡＭに書き込む」**のと同じことです。

したがって私たちは、メモリに順番に並べられたデータをＶＲＡＭにきちんと並べて書き込めばそれでいいのです。



ＶＲＡＭにきちんと長方形に並べるためには、横にいくつ縦にいくつということをはっきりさせなければなりません。

そこで**カウンタ**を使って調べます。

横用カウンタと縦用カウンタの２つが必要です。

そのほかに、現在読み出すべきアドレスと書き込むべきＶＲＡＭのアドレスを示す**レジスタペア**が必要です。

また、データを読み出したり書き込んだりするのに**アキュームレータ**が必要になります。

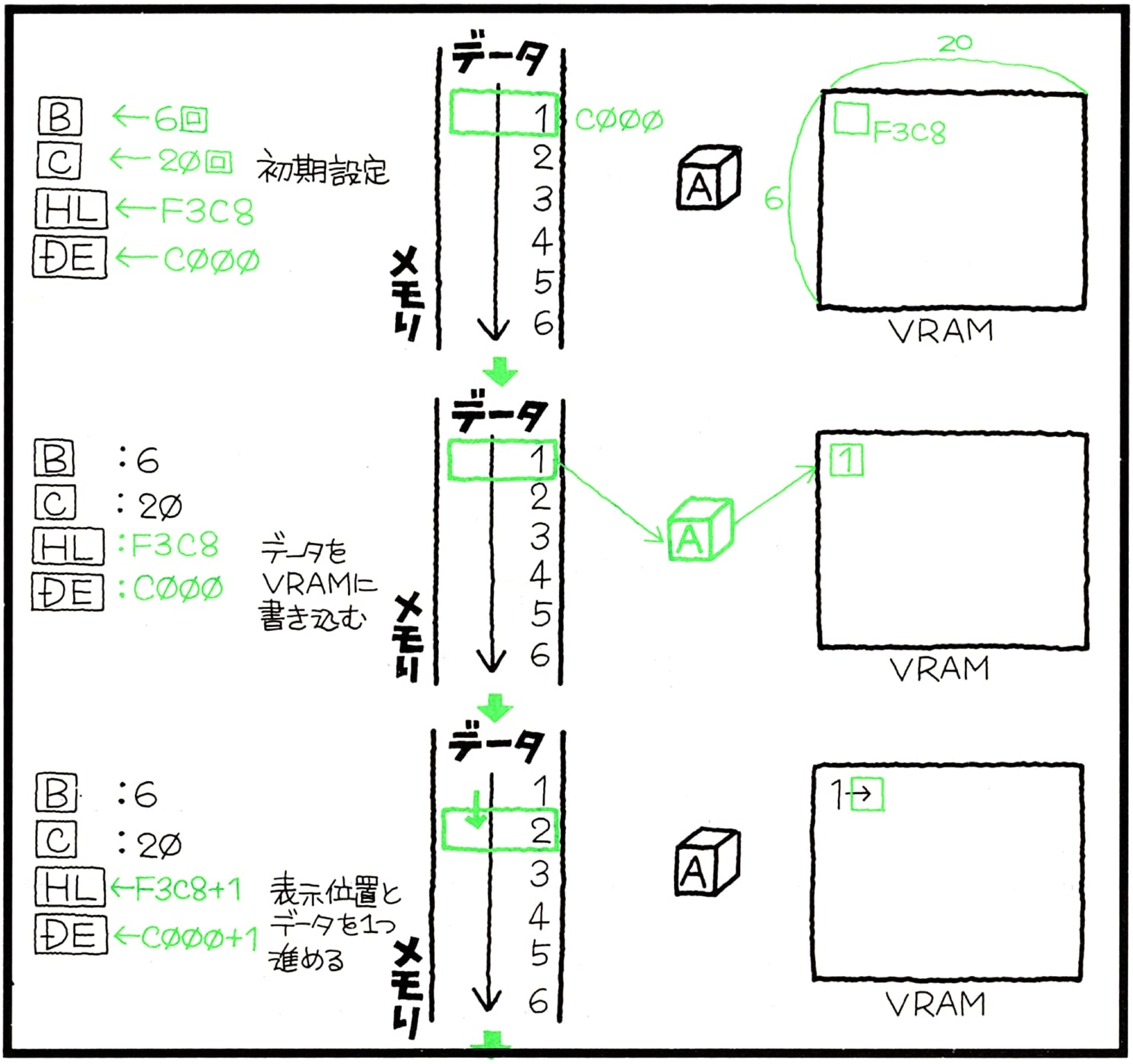
レジスタ表

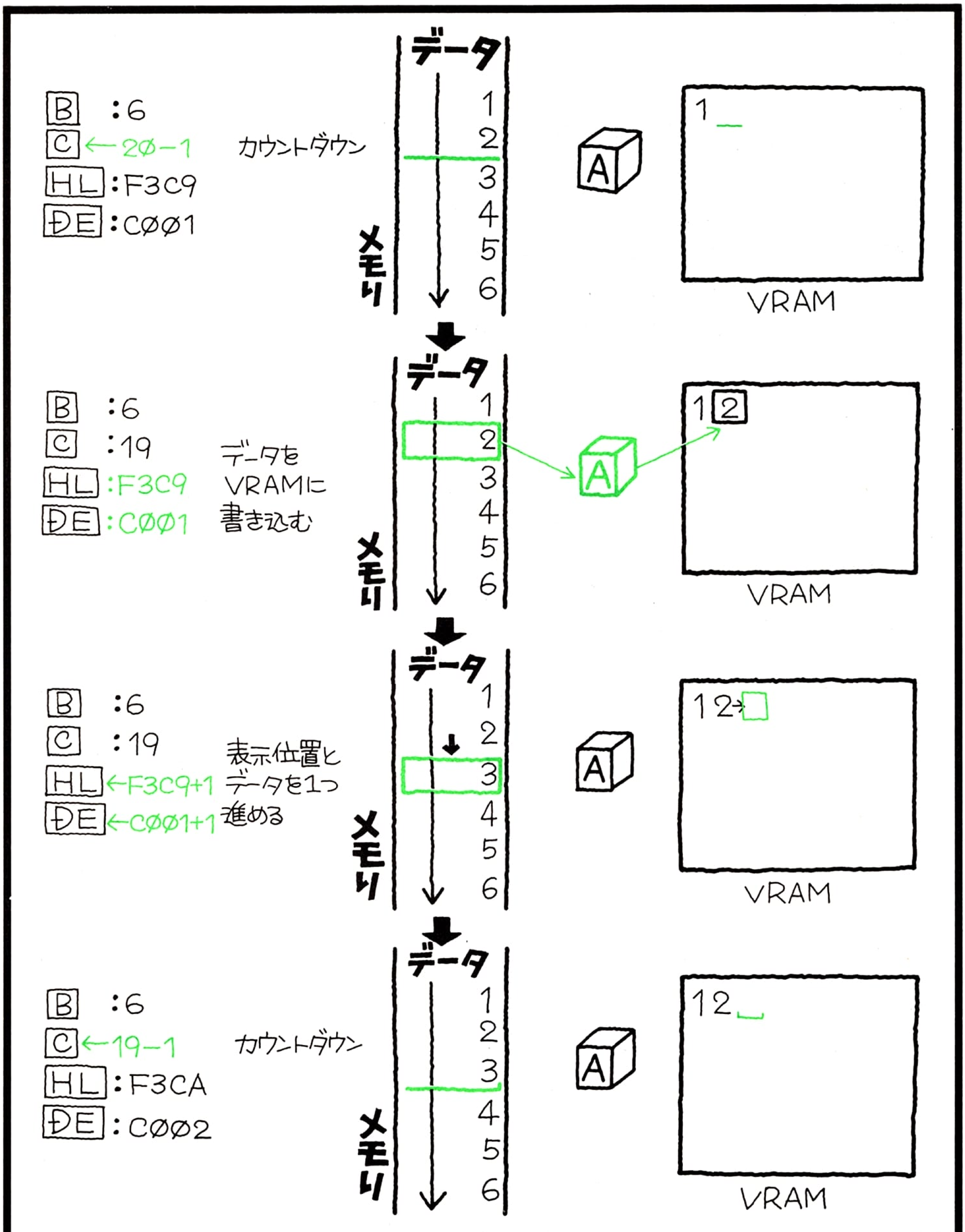
そこで使用するレジスタの表を作ってみます。

レジスタ	役　割
B	縦用カウンタ
C	横用カウンタ
HL	表示アドレス　（書き込むアドレス）
DE	データアドレス（読み出すアドレス）
A	メモリのデータをVRAMに渡す。

こう走らせたい！

実行させた時の事を想像してみます。

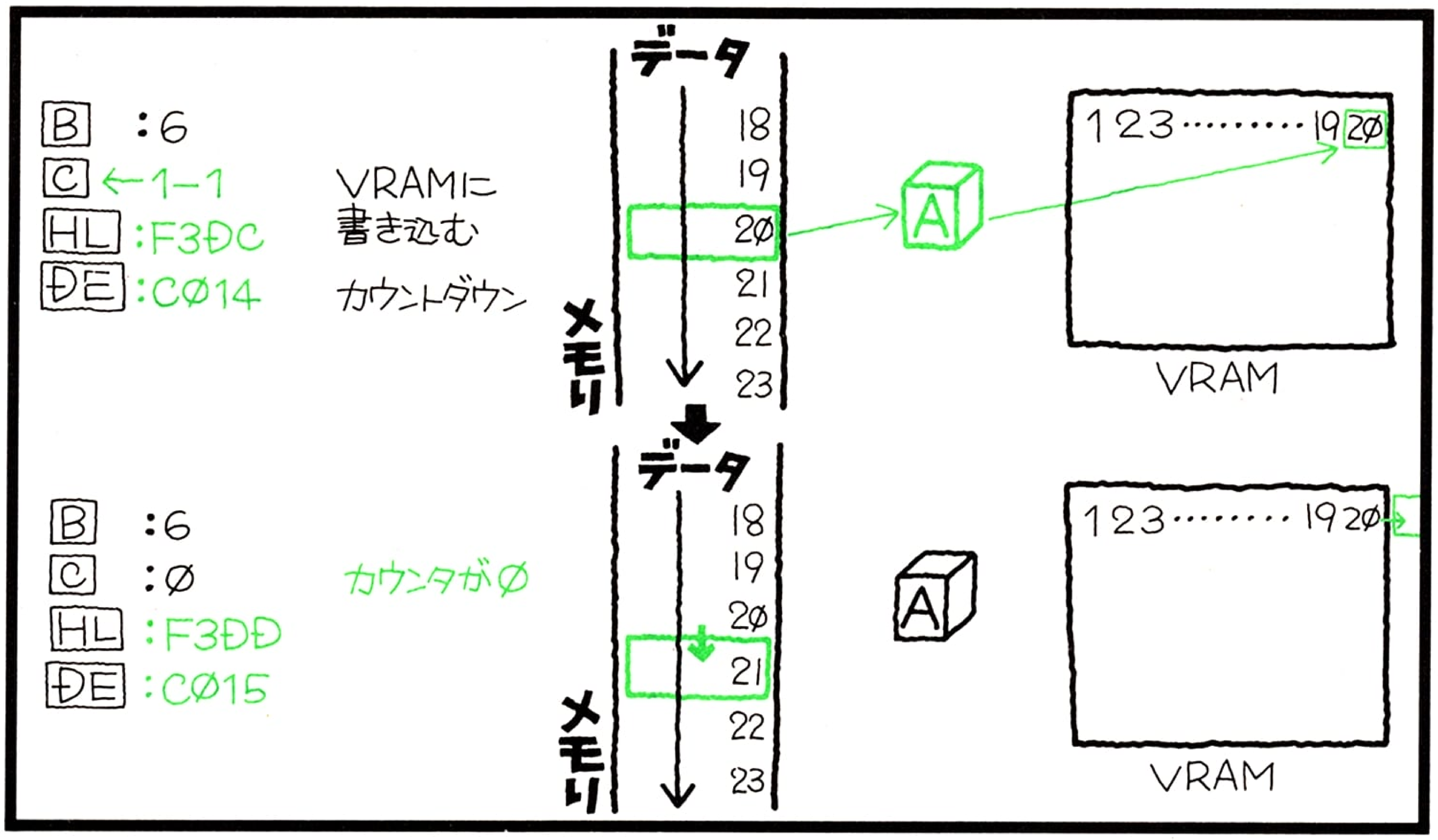




こうやってメモリ上のデータをＶＲＡＭ上に写してゆきます。

メモリからＶＲＡＭへ順番にコピーするわけです。

ただしメモリ上には順番に並んでいてもＶＲＡＭ上には長方形上に並べなければなりません。



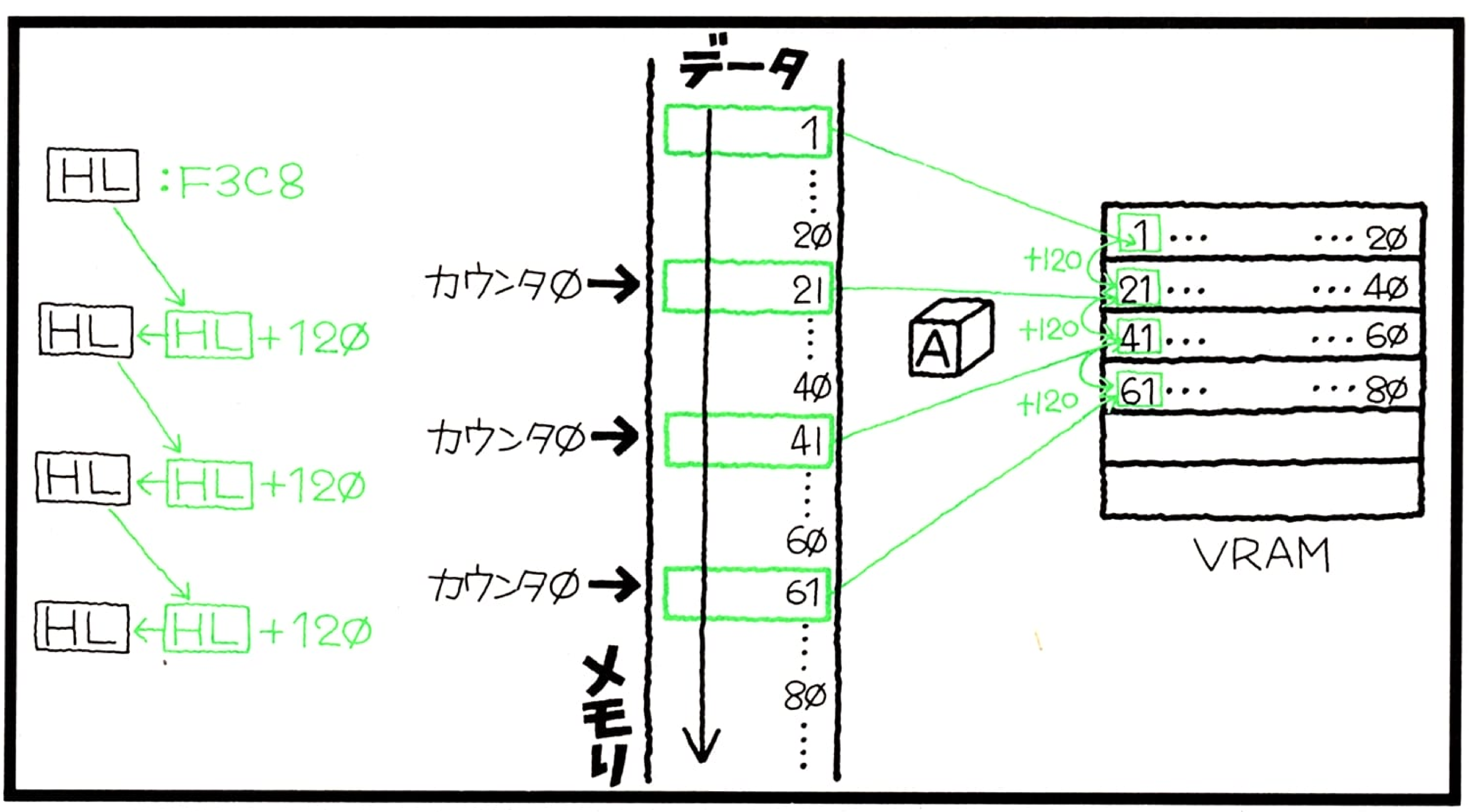
横20コ並べ終わった時、ＤＥレジスタはちゃんと次のデータを指していますが、ＨＬレジスタは長方形からはみ出た所を指しています。

本当なら次の行の先頭アドレスを指しているべきです。

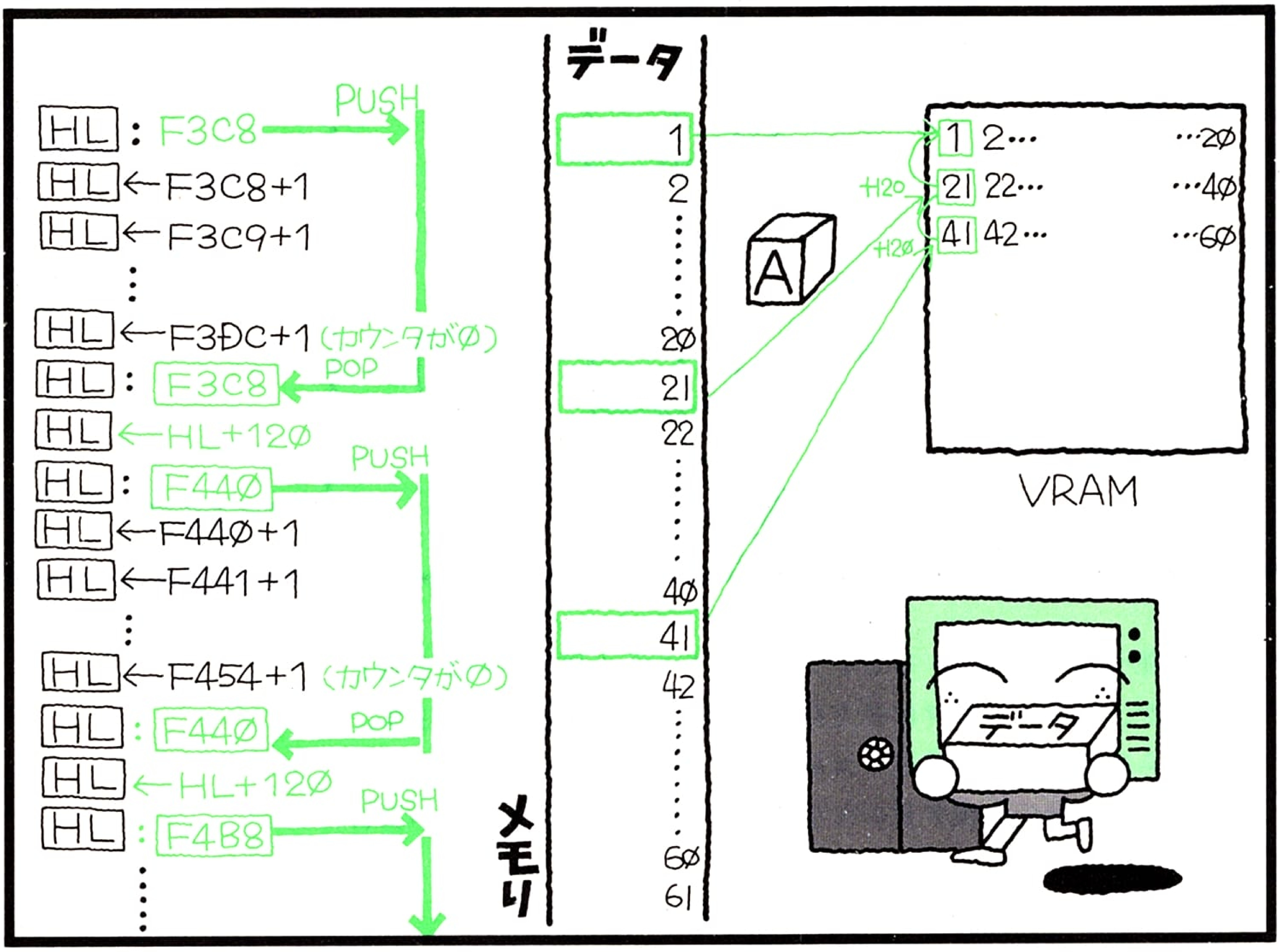
そこで、次の行の先頭アドレスを作ってやらなくてはならないわけです。

この処理は行を変えるごとに行わなければなりません。

20個並べ終えるごとに**前の先頭アドレスに 120 足す**のです。



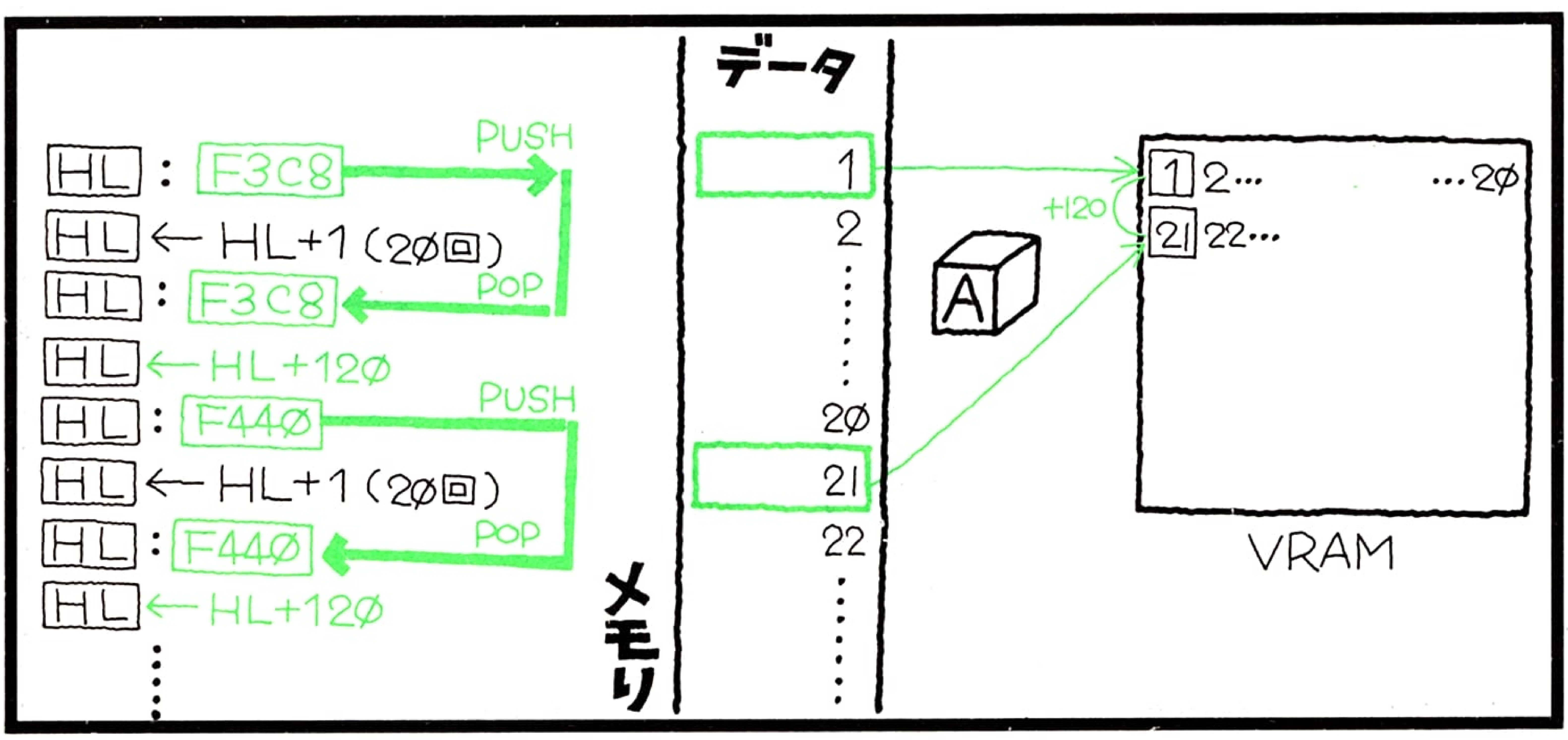
これには「PUSH」「POP」が必要になって来ます。前の先頭アドレスを「PUSH」によって一時保管しておき、カウントが0になって行を変える時に「POP」で呼び出して120を足すのです。



よく見ていただくとわかるのですが、PUSHからPOPまでの間はどれも同じことを20回ずつくり返しています。

そこで**ループ**にしてしまいましょう。

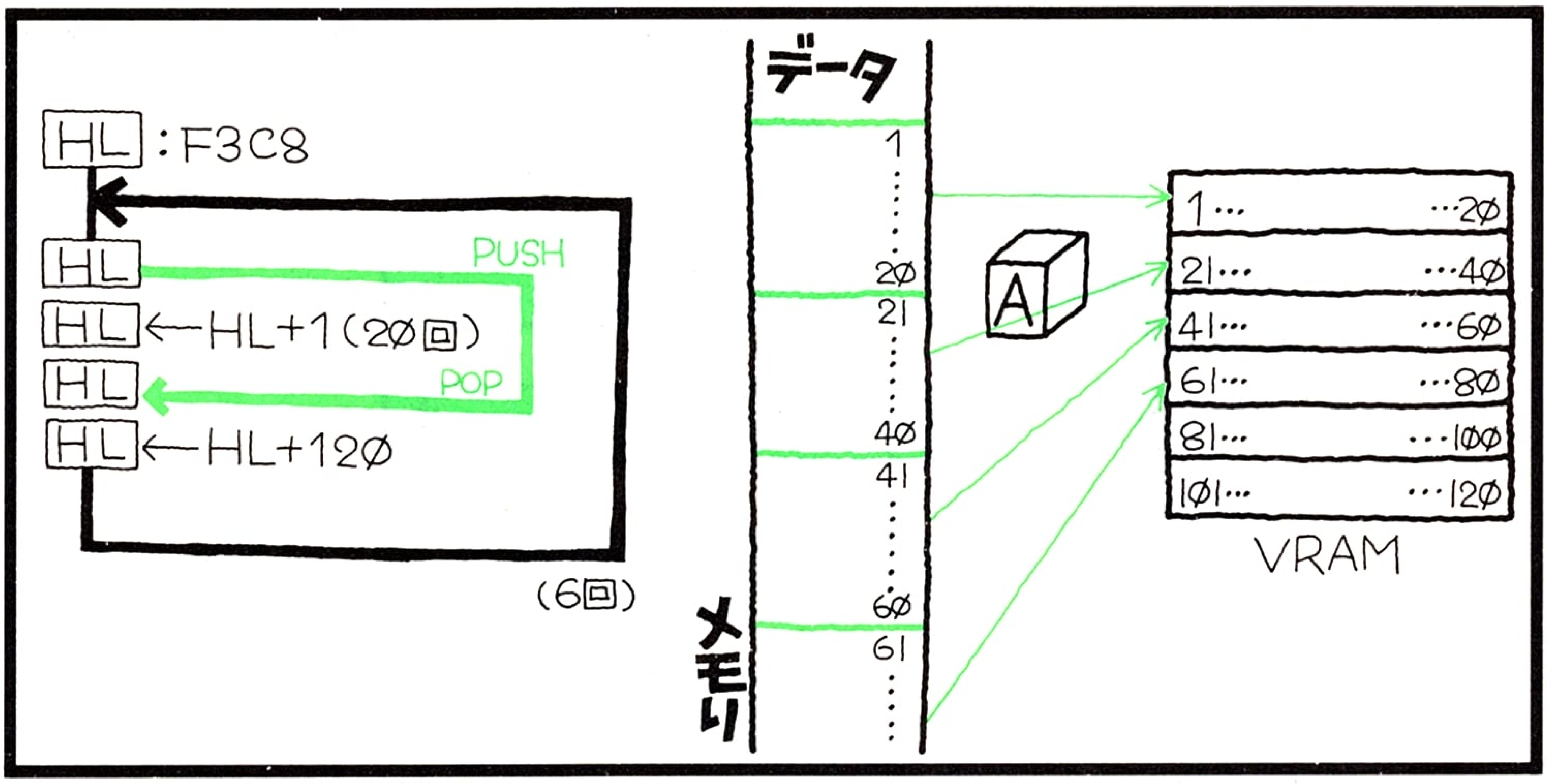
横カウンタがそのままループのカウンタになります。



そこでまた気づかれたと思いますが、このループは1行を表示するプログラムです。

6行くり返すわけですから、つまり**6回のループ**で囲んでしまえばいいわけです。

縦カウンタがこのループのカウンタになります。



ただし、HLに120を足すなんてことは、

ADD HL, レジスタペア

ADD HL, ss

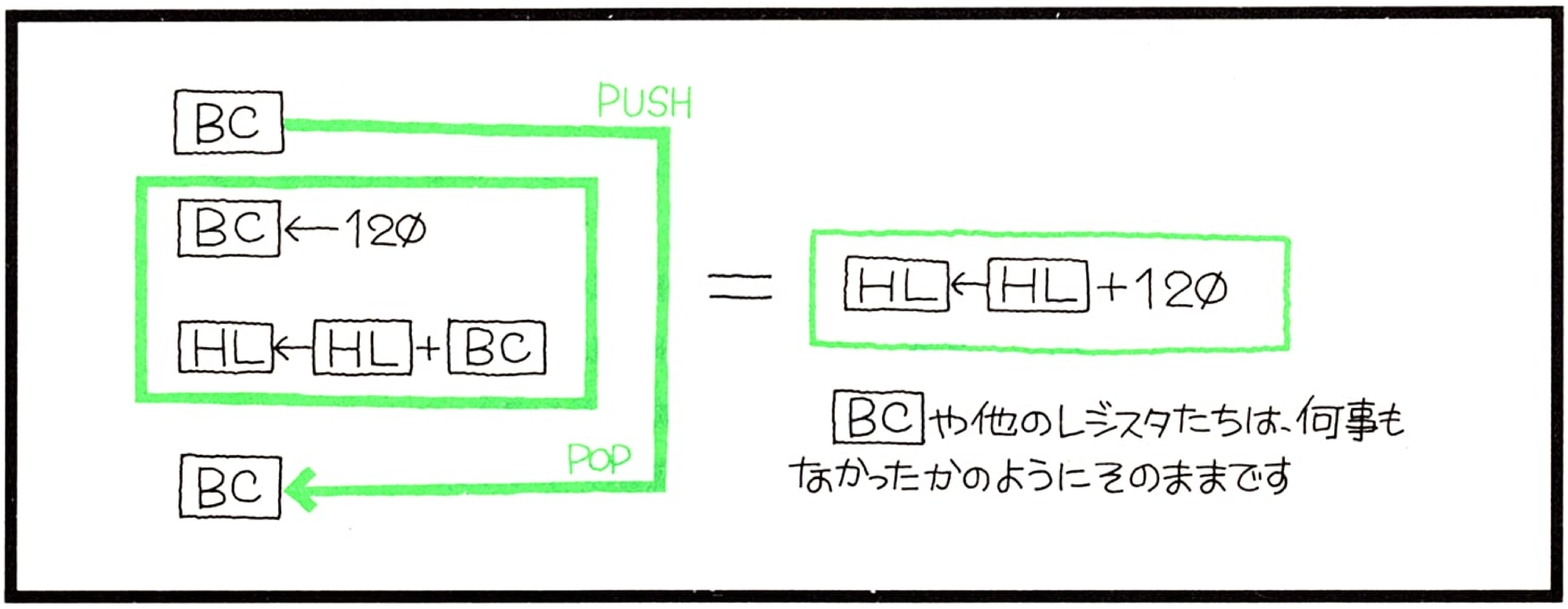
を使わなければなりません。

しかしここではすべてレジスタを使ってしまっているので、レジスタペアをそのままでは使えません。

そこで、また「**PUSH**」と「**POP**」を使うのです。

ここでは、**BCレジスタペア**に120を代入してから加算します。

べつにDEレジスタを使用しても同じことです。

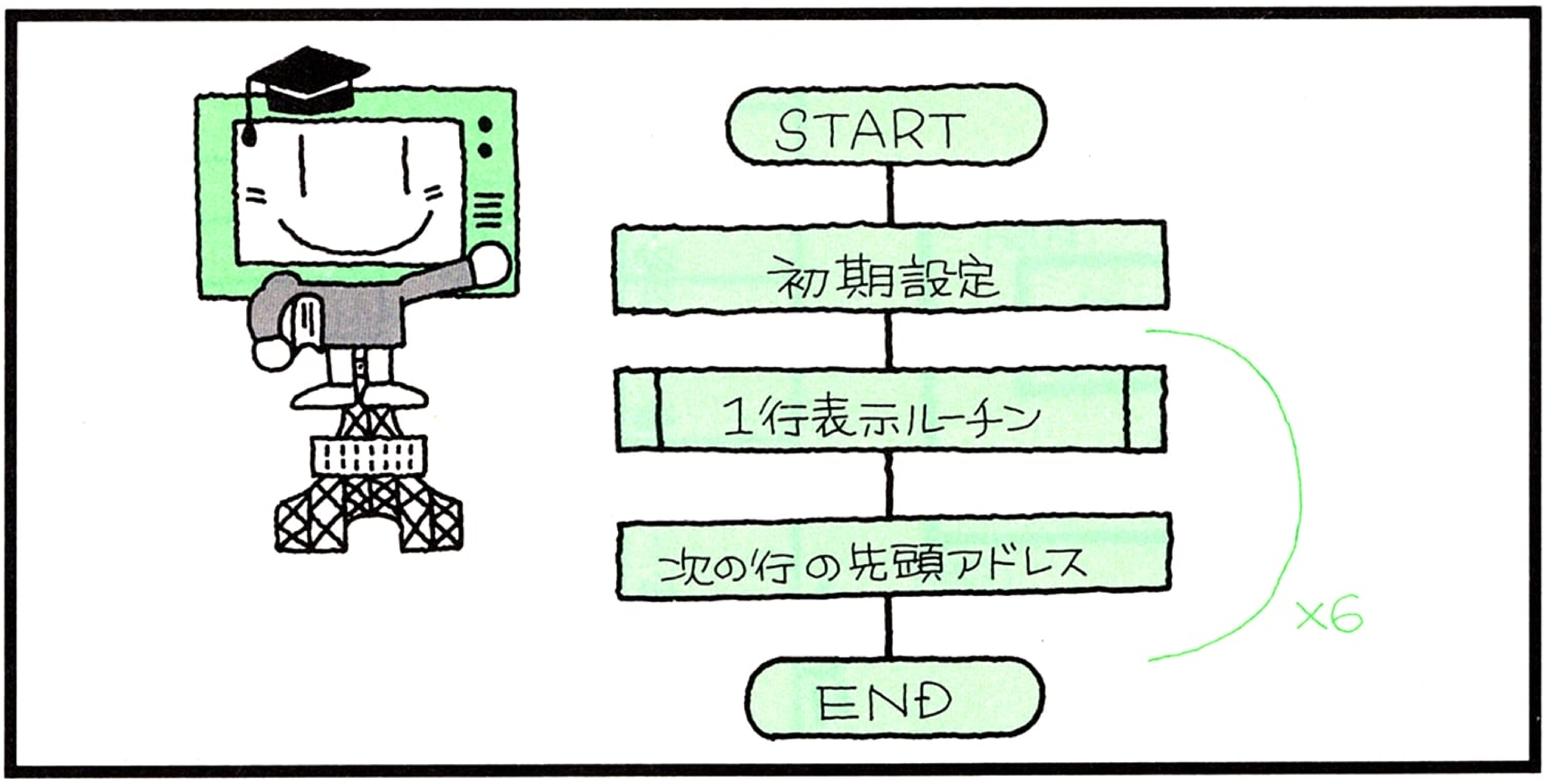


フローチャート

それでは、いよいよフローチャートを書いてみます。

もうほとんど頭の中でわかっているのでずいぶん簡単です。

まずは、全体的な流れ図です。



このうち、**初期設定**とはレジスタの初期値を決めることです。

役割は、

レジスタ	役　割	
B	縦用カウンタ	HL加算用 (120)
C	横用カウンタ	
HL	表示アドレス	
DE	データアドレス	
A	メモリのデータをVRAMに渡す	

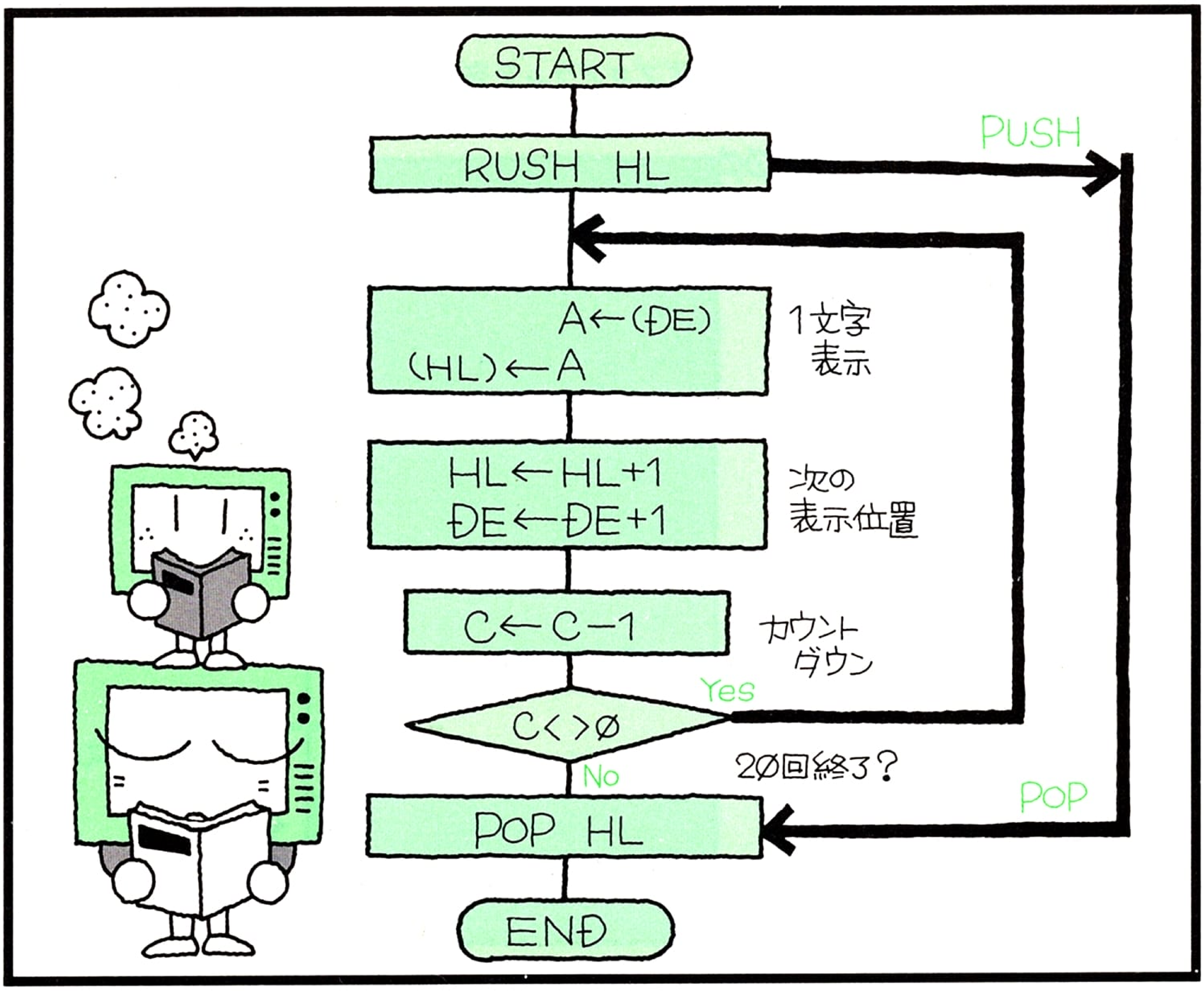
でした。

この中で、最初から決めておかないといけないのは、「B」「C」「HL」「DE」です。

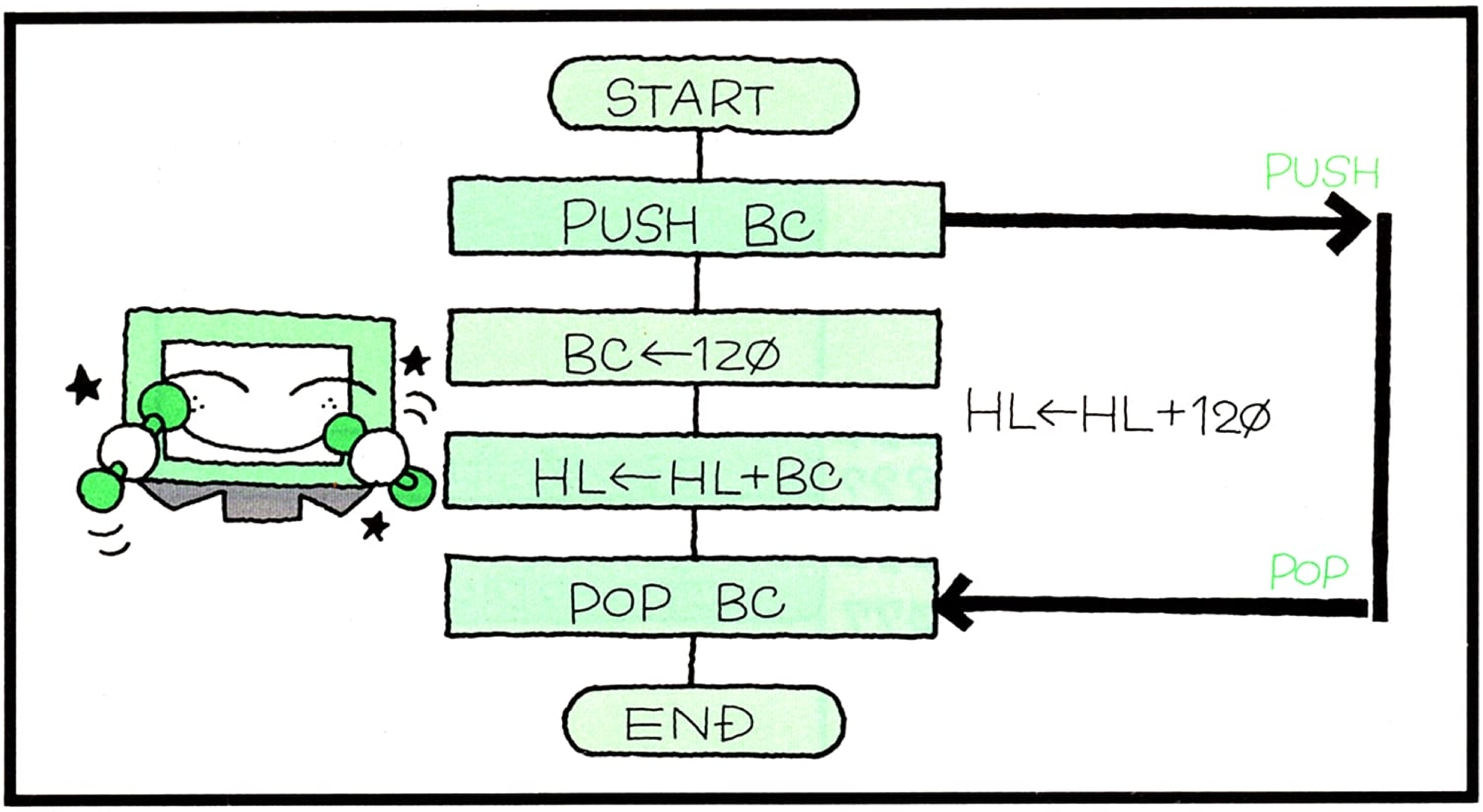
そして「C」だけは毎回セットしなおさなければなりません。

B ：6　　　（カウント6にセット）
C ：20　　（カウント20にセット）
HL：F3C8（表示開始アドレス）
DE：C000（データ先頭アドレス）

1行表示ルーチンは、こうなります。



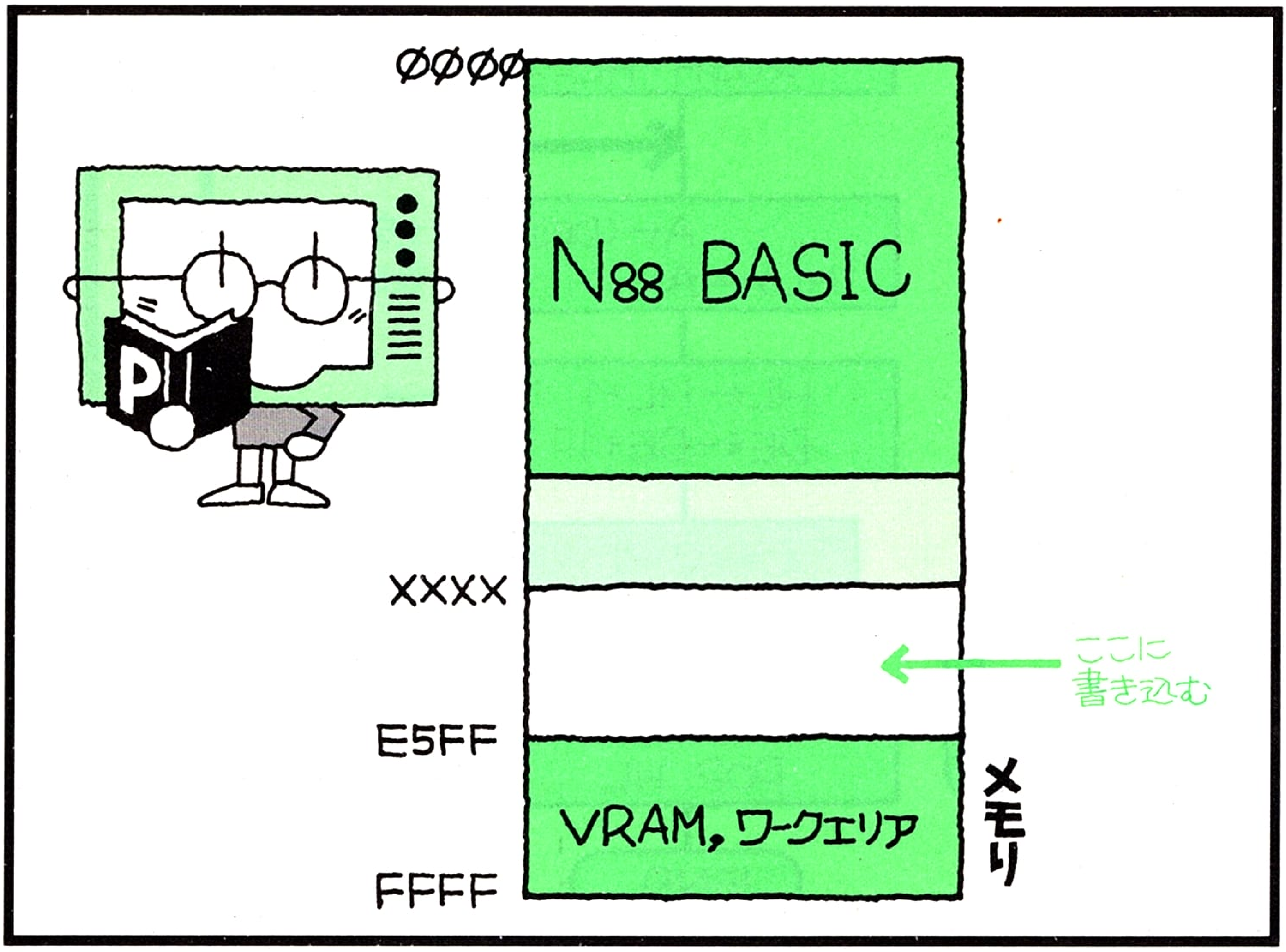
また、**次の先頭アドレスの計算**はこうなります。



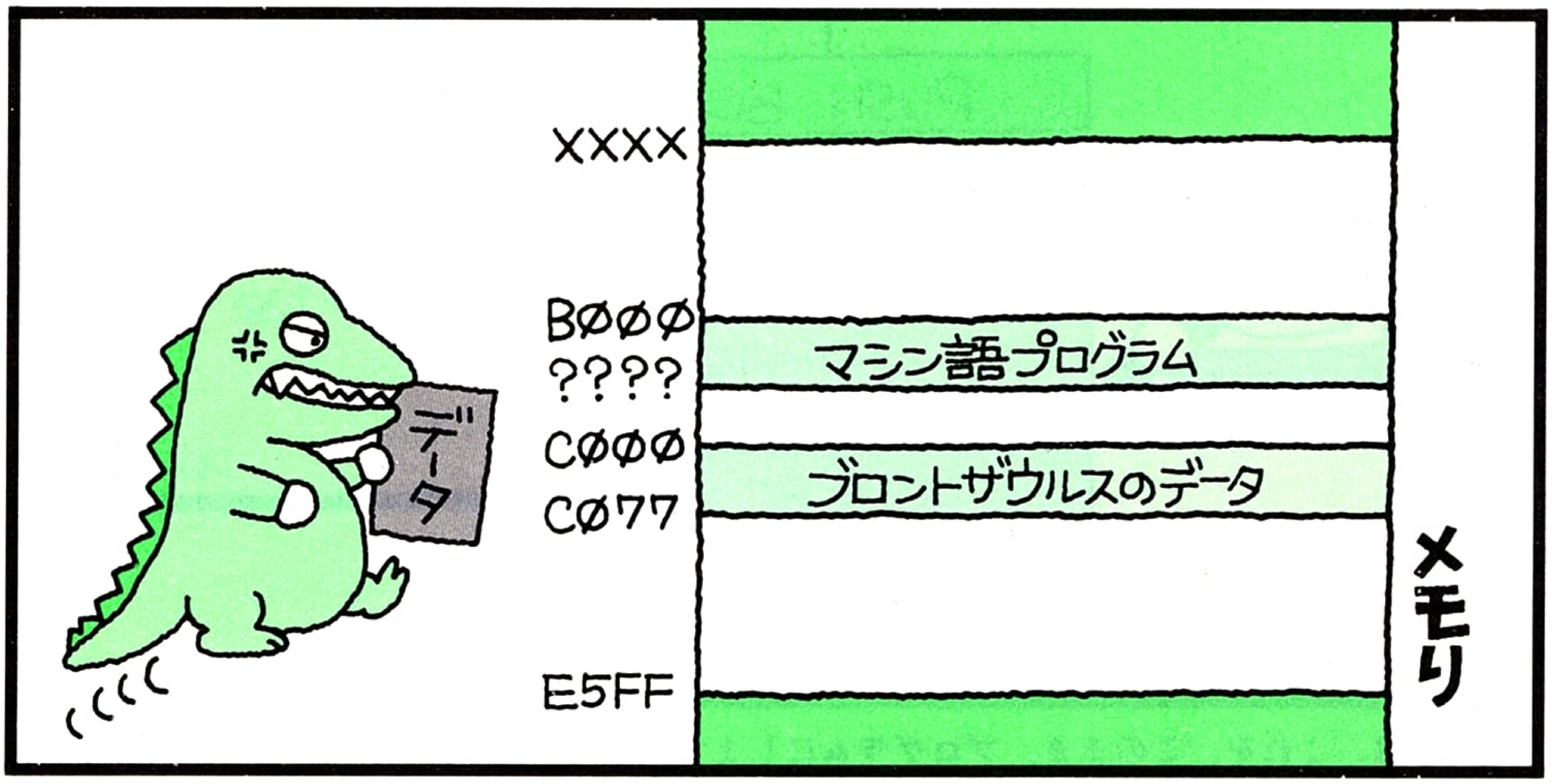
それでは、これを、このまま、**プログラム**にします。

メモリマップ

その前に、まずプログラムを書き込むアドレスをさがします。



ただし、C000〜C077にブロントザウルスのデータが入っている事を考えて、B000から後をプログラムの書き込む場所と考えてみましょう。



プログラム

アドレス	マシン語	ラベル	ニーモニック	コメント
			ORG B000	
B000	21 C8 F3	MAIN	LD HL, F3C8	HL : VRAM
B003	11 00 C0		LD DE, C000	DE : DATA
B006	06 06		LD B, 6	B : Yカウンタ
B008	0E 14	LOOPY	LD C, 14	
B00A	CD 15 B0		CALL LINE1	1行表示!
B00D	CD 20 B0		CALL AD120	先頭アドレス!
B010	05		DEC B	6回ループ
B011	C2 08 B0		JP NZ, LOOPY	
B014	FF		RST 38H	END
B015	E5	LINE1	PUSH HL	
B016	1A	LOOPX	LD A, (DE)	1文字表示
B017	77		LD (HL), A	
B018	23		INC HL	「+1」
B019	13		INC DE	
B01A	0D		DEC C	20回ループ
B01B	C2 16 B0		JP NZ, LOOPX	
B01E	E1		POP HL	
B01F	C9		RET	リターン
B020	C5	AD120	PUSH BC	
B021	01 78 00		LD BC, 0078	HL←HL+120
B024	09		ADD HL, BC	
B025	C1		POP BC	
B026	C9		RET	リターン

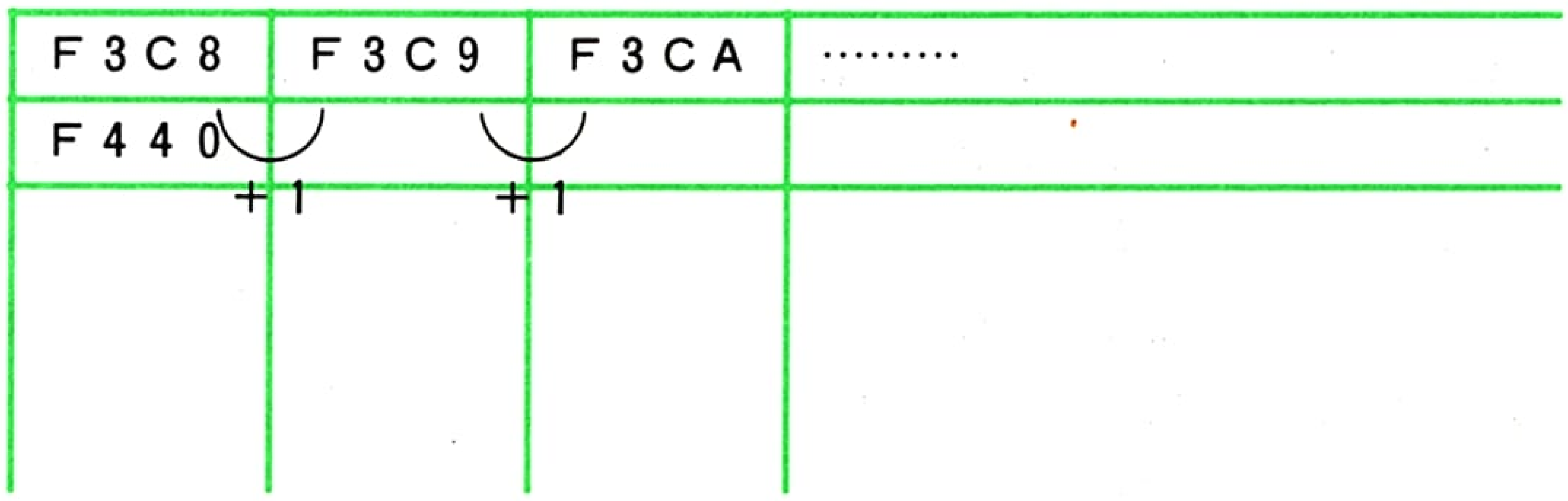
こういうプログラムになるわけです。

実行!

実はこれは、80字モード用で作りました。

表示アドレスを示すHLレジスタに「＋１」ずつ足してとなりの番地を指していたからです。

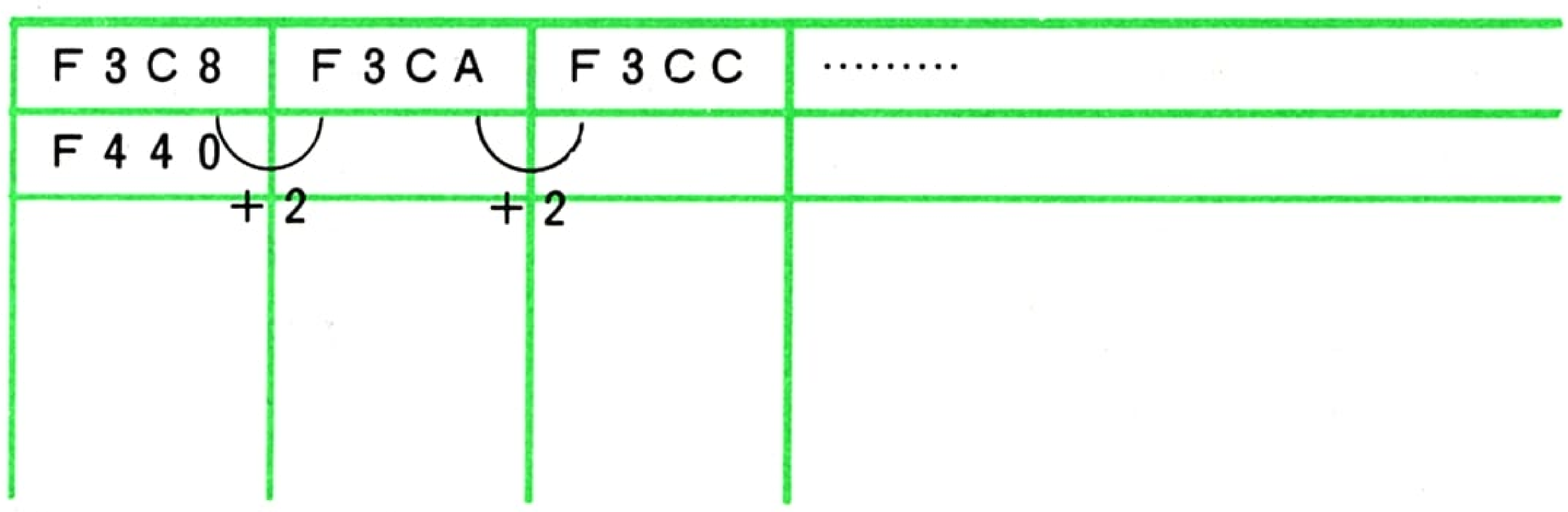
VRAM（80字モード）



40字モードなら「＋２」しなければならなかったのです。

画面に表示されるＶＲＡＭが40字モードでは２つとびになっているからです。

VRAM（40字モード）



とにかく、プログラムを入力してみましょう。

プログラムの部分と、データの部分です。

```
B000 21 C8 F3 11 00 C0 06 06 0E 14 CD 15 B0 CD 20 B0
B010 05 C2 08 B0 FF E5 1A 77 23 13 0D C2 16 B0 E1 C9
B020 C5 01 78 00 09 C1 C9
```

```
C000 87 87 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C010 00 00 00 00 00 87 00 00 00 80 80 80 80 00 00 00
C020 00 00 00 00 00 00 00 00 00 87 87 87 87 87 87 87
C030 87 87 00 00 00 00 00 00 00 00 00 00 00 00 00 87
C040 87 87 87 87 87 87 E5 00 00 00 00 00 00 00 00 00
C050 00 00 00 87 87 87 87 87 87 87 87 E5 00 00 00 00
C060 00 00 00 00 00 00 00 87 00 00 94 94 00 87 E6 87
C070 86 85 84 83 82 81 80 80
```

80字モードですから、ＢＡＳＩＣで、

```
WIDTH 80,25 ⏎
```

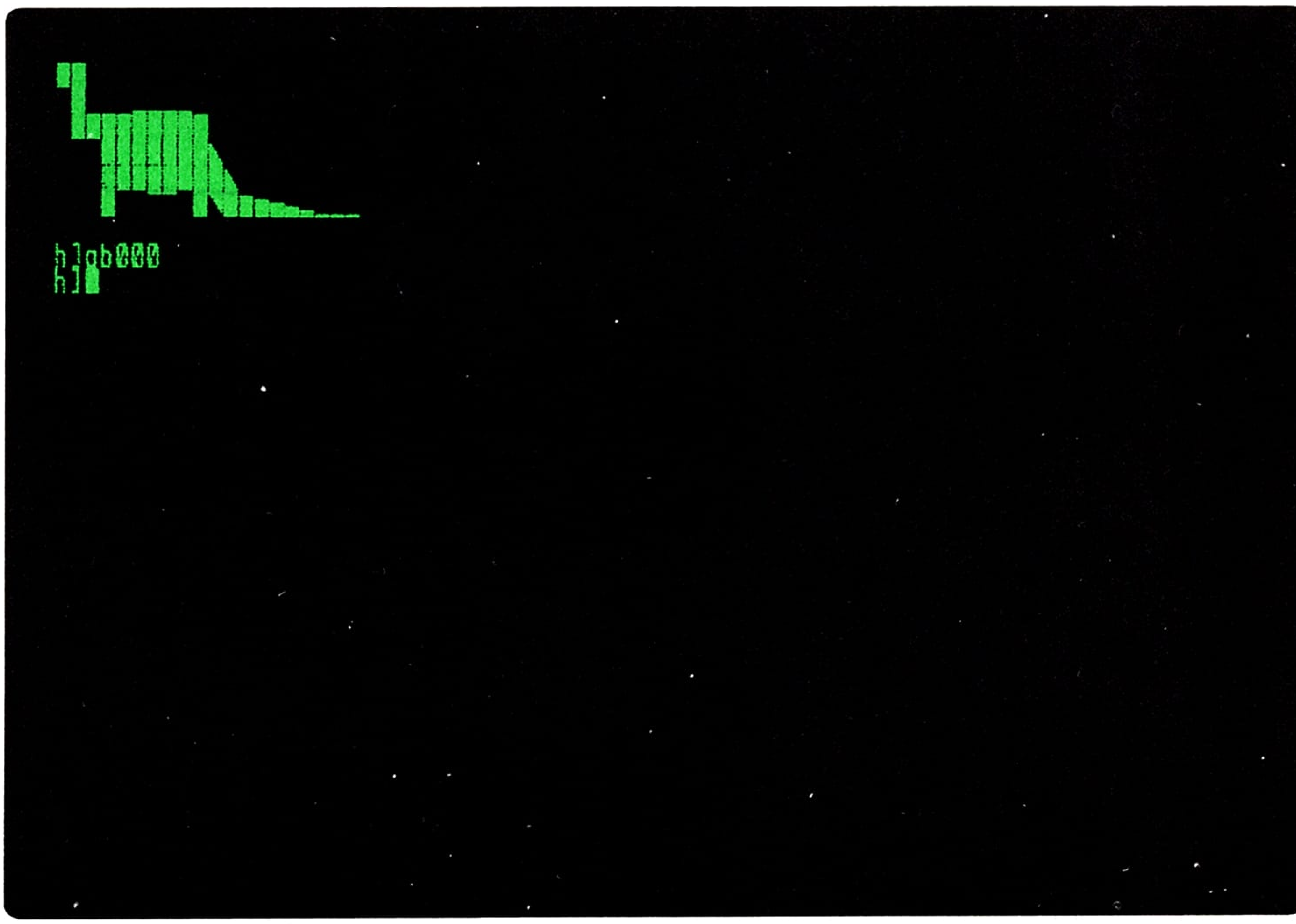
としてから、**マシン語モード**に戻り、

```
MON ⏎
```

Gコマンドで、

```
h]gb000 ⏎
```

実行させましょう！



ブロントザウルスが現われました。

5·2　ステゴザウルス

今度はステゴザウルスです。



こんなステゴザウルスを表示するプログラムを作ってみましょう。

目標は、

ステゴザウルスを表示させる

ことです。

ただし、今度は**「どこに表示するか」**ということをこちらから指定できるようにしてみましょう。

指定された場所にステゴザウルスを表示する

手っ取り早くするために、表示位置の入力はＢＡＳＩＣで行うことにします。そして、そのあとでマシン語に飛んでステゴザウルスを出現させるのです。

つまり、**ＢＡＳＩＣ＋マシン語のプログラム**を作ろうとしているわけです。

表示位置を入力する（ＢＡＳＩＣ）

↓

ＢＡＳＩＣからデータを受け取る（マシン語）

↓

ステゴザウルスを表示させる（マシン語）

ＢＡＳＩＣ部

まずは、ＢＡＳＩＣ部から作っていくことにします。

ここでは、

表示位置のＸ，Ｙデータを入力して

マシン語にデータを送る

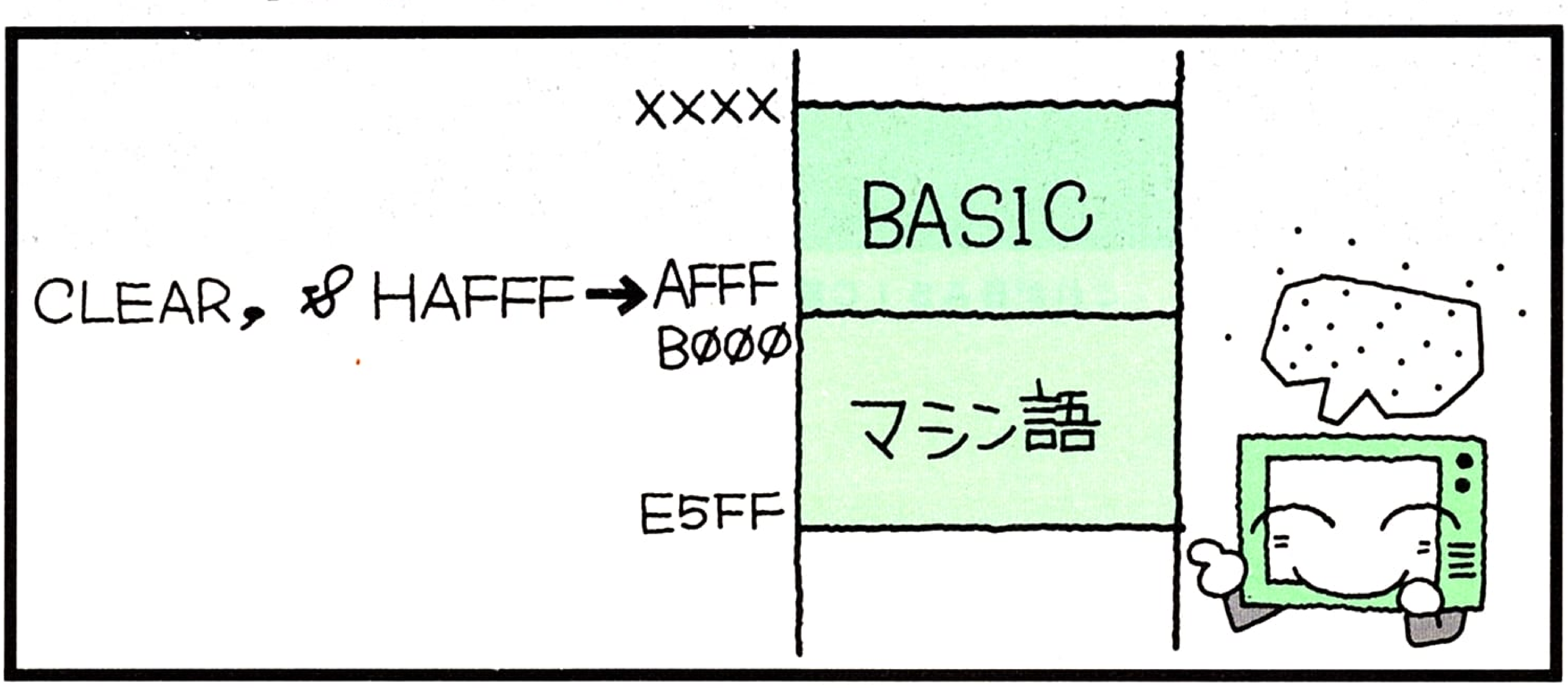
ということをするわけです。

データの入力は、「ＩＮＰＵＴ」で入力すればいいわけです。

マシン語にデータを送るのは「Ａ＝ＵＳＲ(０)」の丸カッコの中に入れて送ることにしましょう。

それではすぐプログラムを作ったらいいかのように思いますが、実はその前にしておかなくてはならない事があります。
マシン語プログラムを入れておく領域を決めておかないといけないのです。

いつものように、Ｂ０００までをＢＡＳＩＣ、それ以降をマシン語の領域にしてみます。
「ＣＬＥＡＲ」文で領域をわけるのです。



プログラム

これが、

ＣＬＥＡＲでＢＡＳＩＣ領域とマシン語領域をわける
表示位置のＸ，Ｙデータを入力して
マシン語にデータを送る

これから作ろうとしているプログラムです。

マシン語に２つのデータを送らなくてはならないのですから、

ＵＳＲ関数

A=USR(X*256+Y)

を使います。

そのほかに、変数を整数型に宣言したり、マシン語をＢ０００から呼び出すことを決めておくことも必要です。

```
10 CLEAR ,&HAFFF
20 DEFINT A-Z
30 DEF USR=&HB000
40 INPUT "X=",X
50 INPUT "Y=",Y
60 A=USR(X*256+Y)
70 END
```

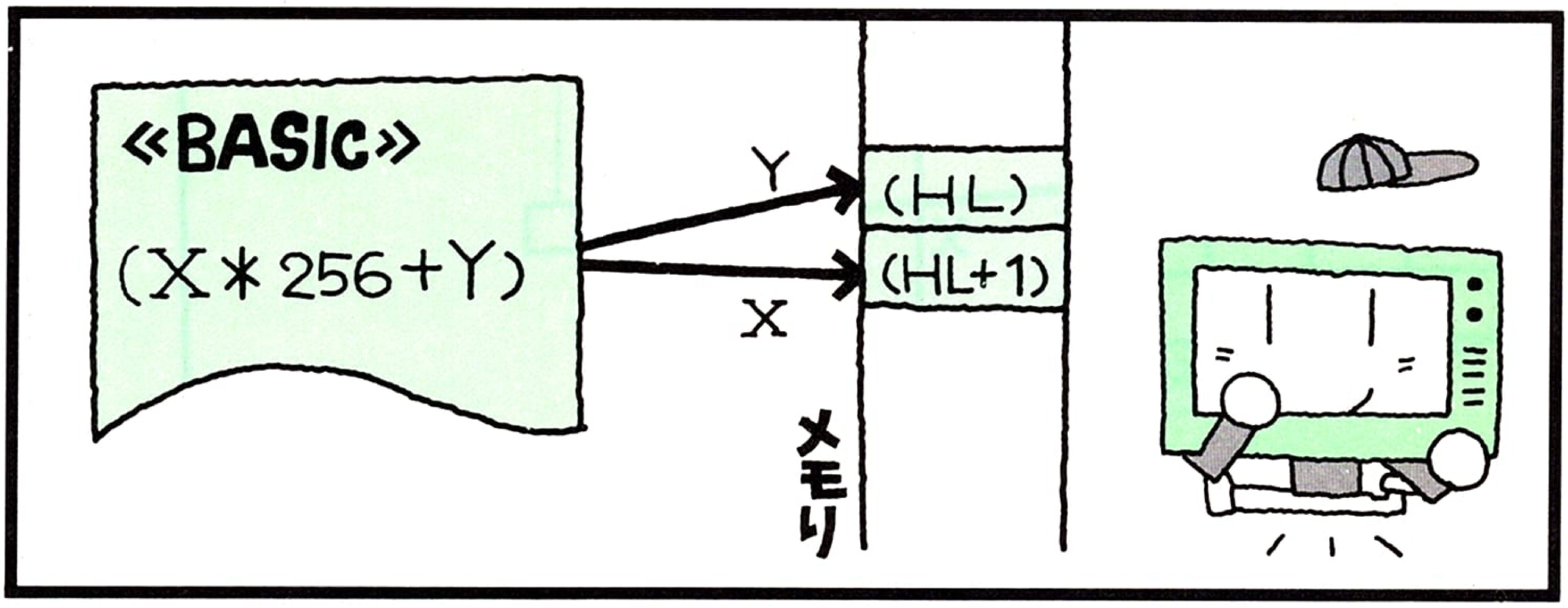
これがＢＡＳＩＣ部のプログラムです。

データ解読

それでは今度はマシン語部を作っていくことにしましょう。

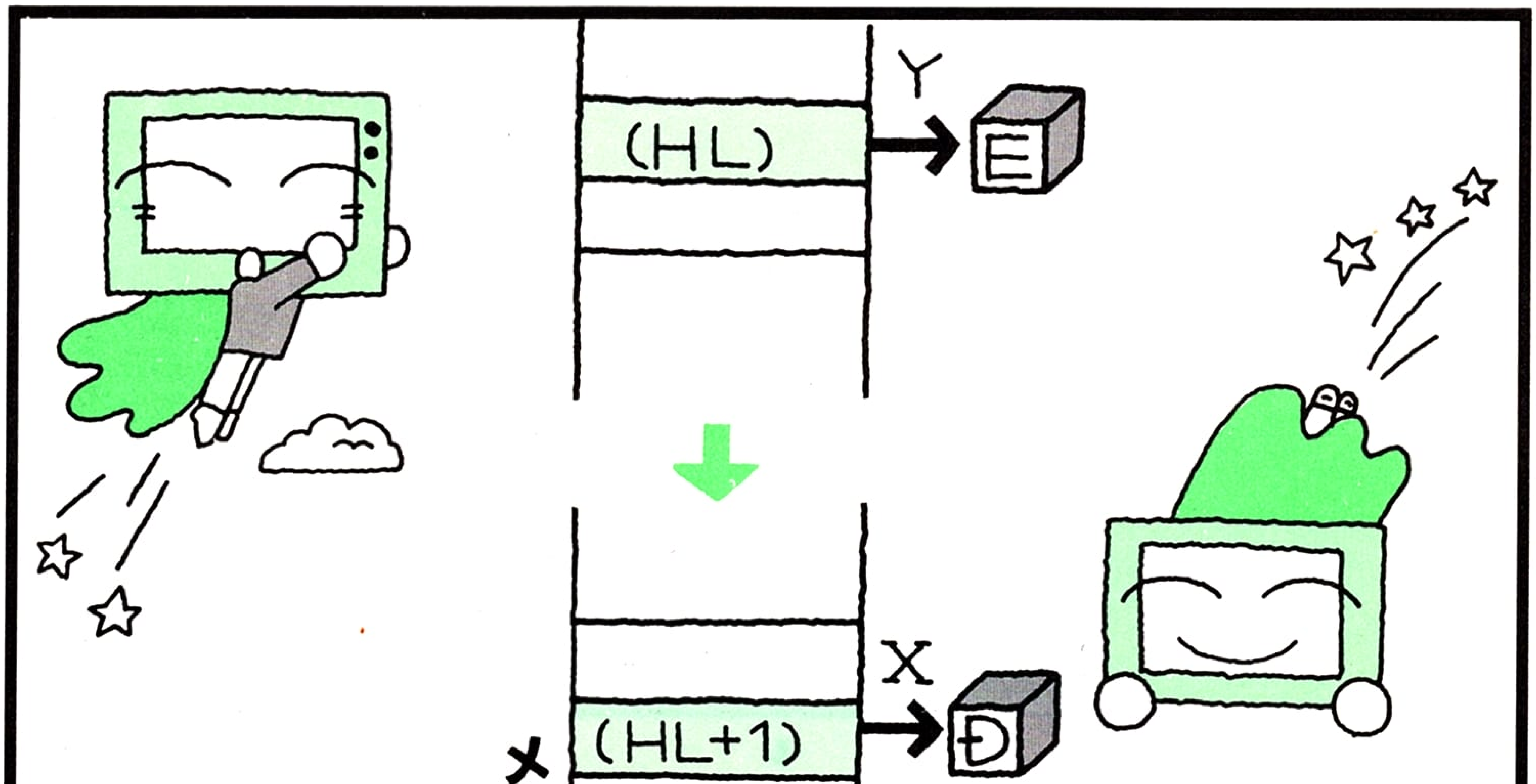
まず最初は、ＢＡＳＩＣから送られて来たデータを読み取るプログラムです。

ＢＡＳＩＣから送られて来たデータは、ＨＬレジスタが示すアドレスとその次のアドレスに書き込まれています。



メモリからレジスタに順番に読み出してやりましょう。

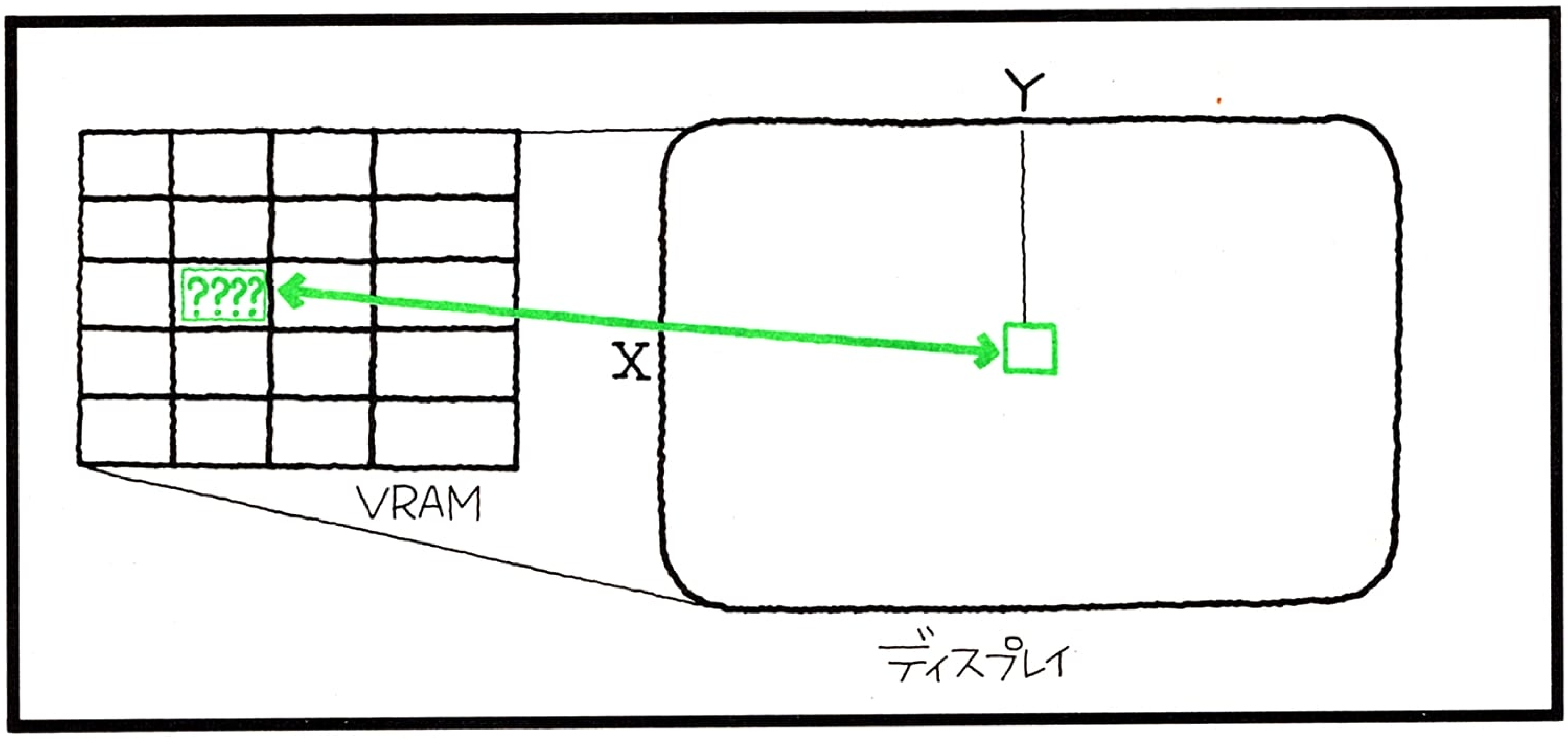
ただし、ＨＬレジスタはアドレスを指定するのに使われているのでその他のレジスタに読み出します。



ここでは、あとで便利なことがあるので**DEレジスタ**に読み出すことにします。

これで、DEレジスタにXY成分が入ったわけですが、これだけではまだ不十分です。

なぜなら、XY成分が示す場所に表示するためには、その場所に対応する**VRAMのアドレスが必要**だからです。



X，Yデータから VRAMアドレスを求めるのは

429D

をCALLすることによって計算できます。

HLにXYのデータを入れてCALLすると、HLレジスタペアに VRAMアドレスが入力されるのです。

H：X座標＋1
L：Y座標＋1

⬇

429D
（VRAMアドレス計算）

⬇

HL：VRAMアドレス

そこで、HLレジスタにXY成分を移しかえることにしましょう。

そうすれば、429DをCALLするだけでVRAMアドレスを求

めることができるからです。

では、

```
LD   HL, DE
```

としてＨＬにＤＥを代入しましょうか。

実はこういう命令は存在しないのです。
この場合なら、

```
LD   H, D
LD   L, E
```

とでもしなければならないのです。

プログラムを作っていく時、「こんな命令はあるだろう」と思っていても実は存在しなかったということがあります。
命令を使う時は一度確かめるようにしましょう。

ところで、ＨＬにＤＥの値を入れたい場合、ＨＬとＤＥの値を交換してもできることが考えられます。
そこで、そんな命令を使ってみることにします。

EX DE, HL

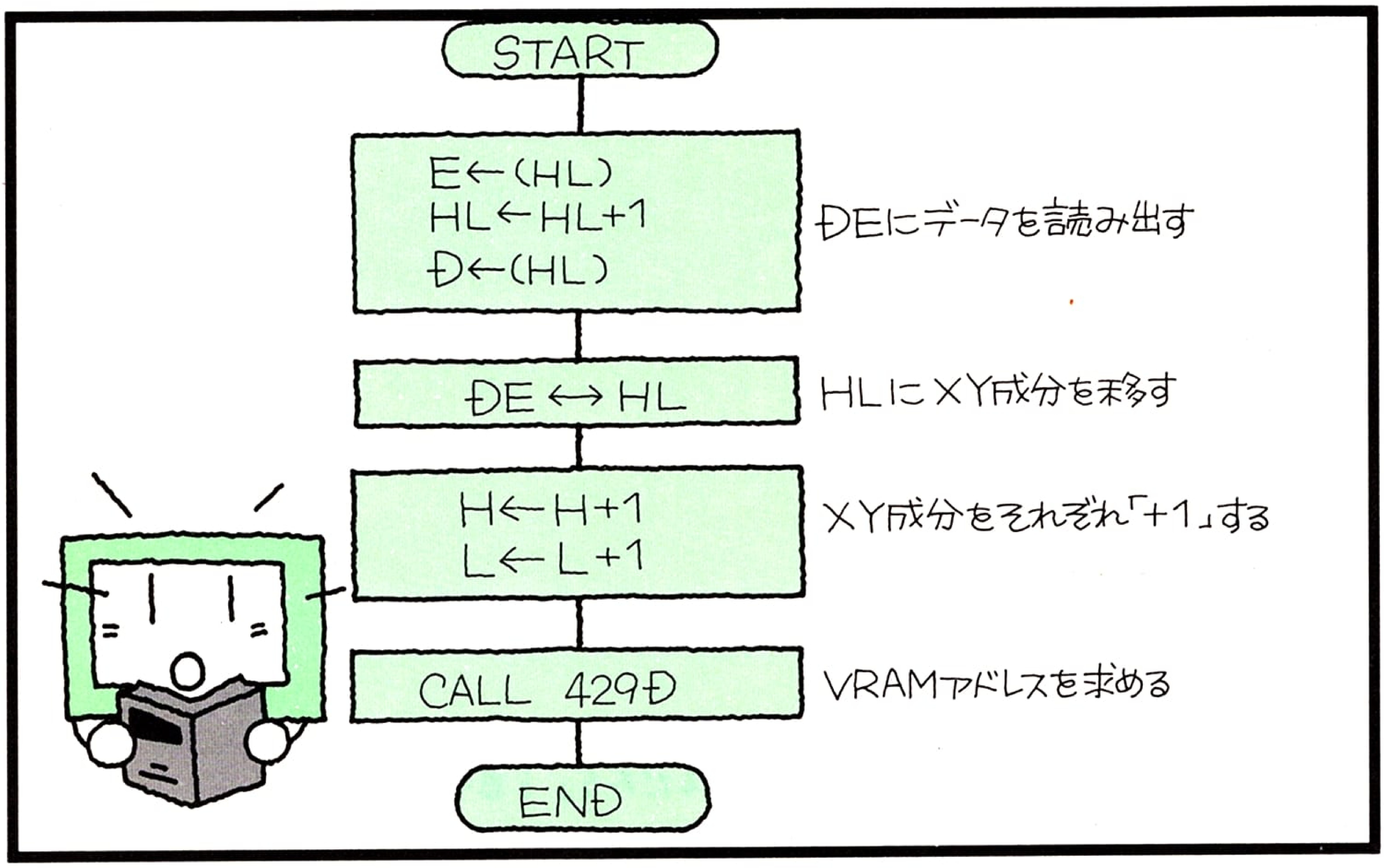
ＥＸ　ＤＥ，ＨＬ
（エクスチェンジ）

これは、**ＤＥレジスタとＨＬレジスタの中身を交換する命令**なのです。
すなわち、

ＨＬレジスタの値がＤＥへ
ＤＥレジスタの値がＨＬへ

代入されるわけです。
これを１回使うことによって、ＤＥレジスタに入っているＸＹ成分がＨＬレジスタに移しかえられるのです。

あとは、Ｈレジスタ、Ｌレジスタそれぞれに１を足して「４２９Ｄ」をＣＡＬＬすれば表示位置が求まります。



プログラムはこうなります。

アドレス	マシン語	ラベル	ニーモニック	コメント
			ORG B000	START
B000	5E	START	LD E, (HL)	E←(HL)
B001	23		INC HL	
B002	56		LD D, (HL)	D←(HL)
B003	EB		EX DE, HL	DE↔HL
B004	24		INC H	「+1」
B005	2C		INC L	
B006	CD 9D 42		CALL 429D	VRAM計算

これでＨＬレジスタに表示アドレスが入るわけです。

イメージを浮かべる

残るはステゴザウルスを表示させるプログラムです。

これはもうブロントザウルスの時とまったく同じですから簡単ですね。

今度は40字モードで表示させてみましょうか。

まず、



を、7×3の長方形の絵と考え、**そのデータを作ります。**



これを**アスキーコードに変換**して、



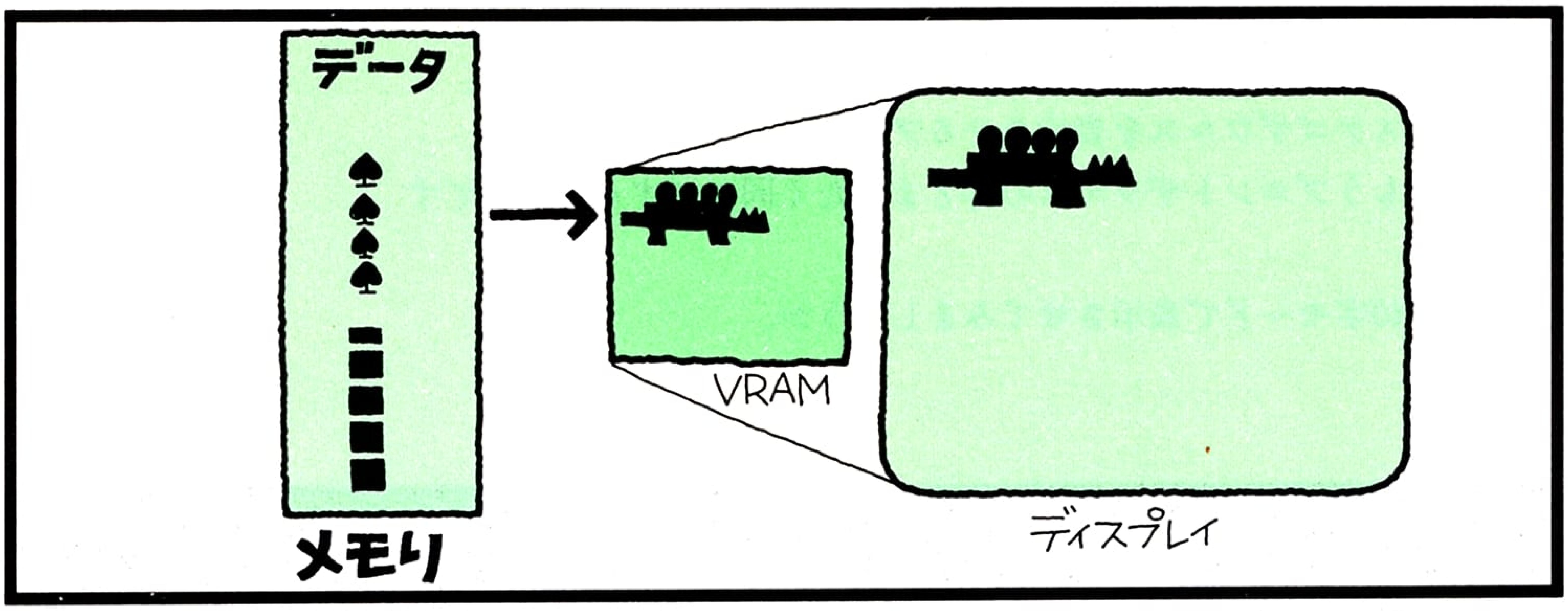
メモリに書き込んで、



データにします。

あとはHLで指定されたアドレスから順番に長方形になるように、データをコピーすればいいわけです。

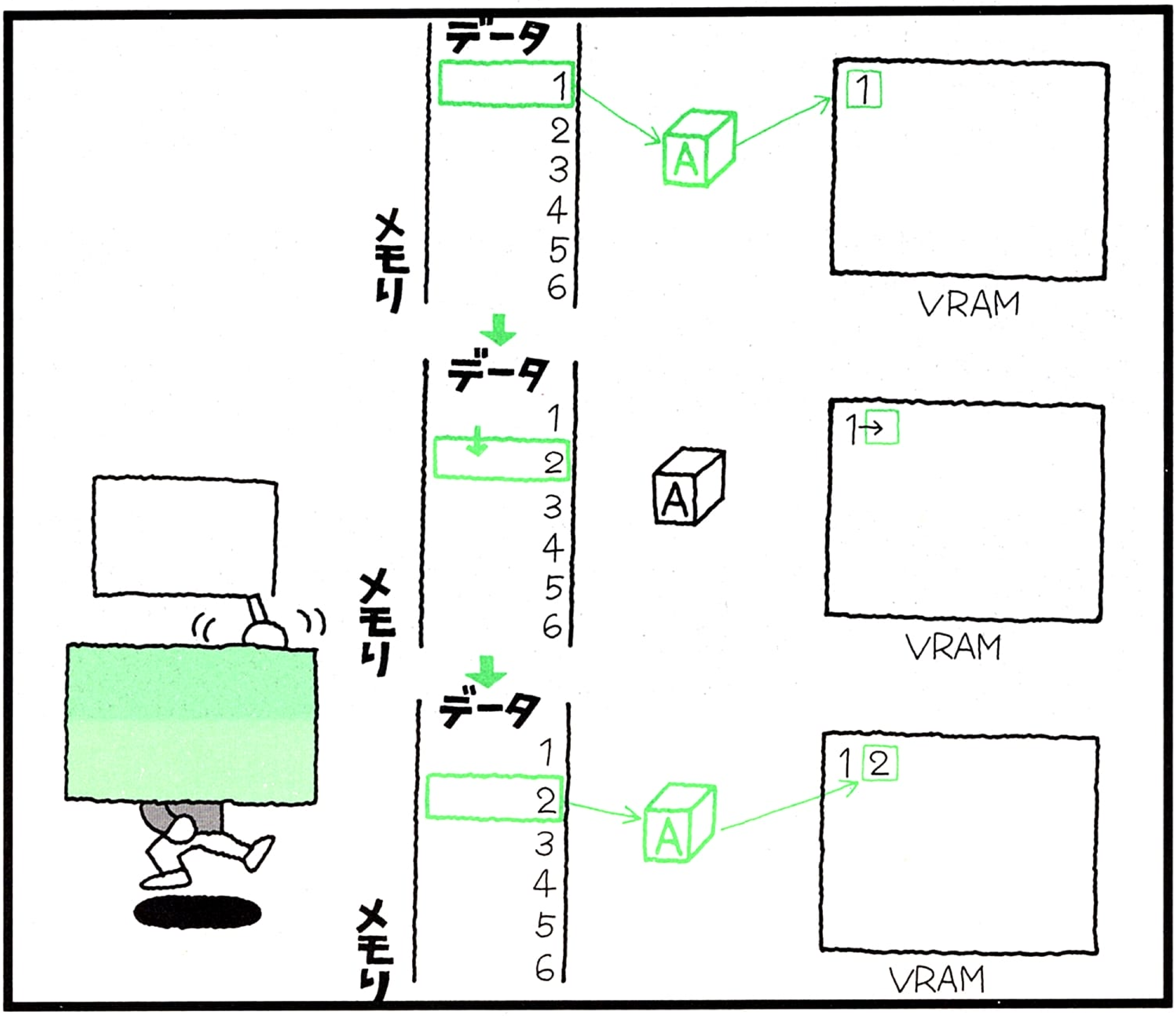
その過程を思い浮かべてみましょう。

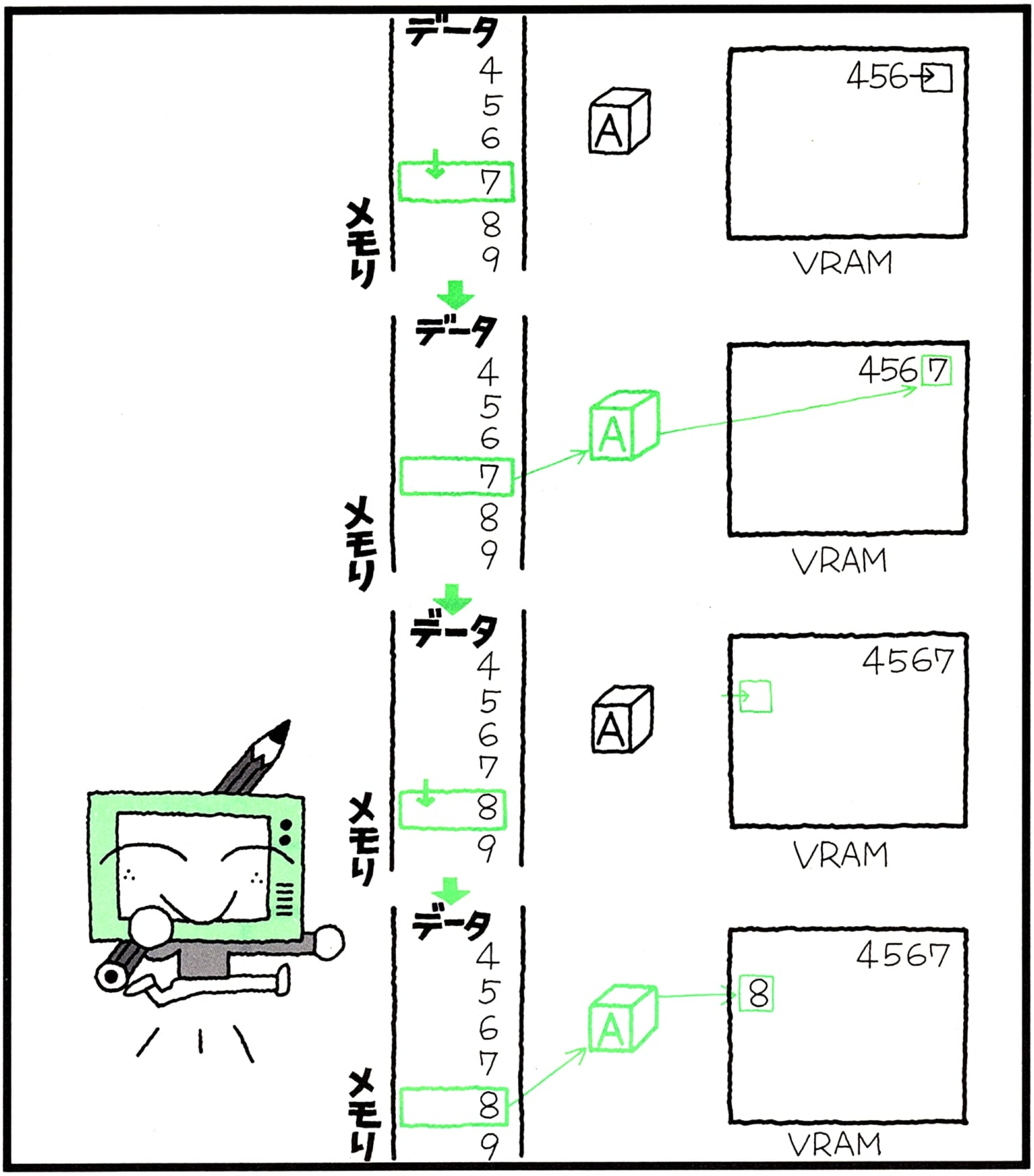


こう走らせたい！

まず、横に７、縦に３の長方形にすることを２つのカウンタに覚え込ませます。

それからコピーをしていきましょう。



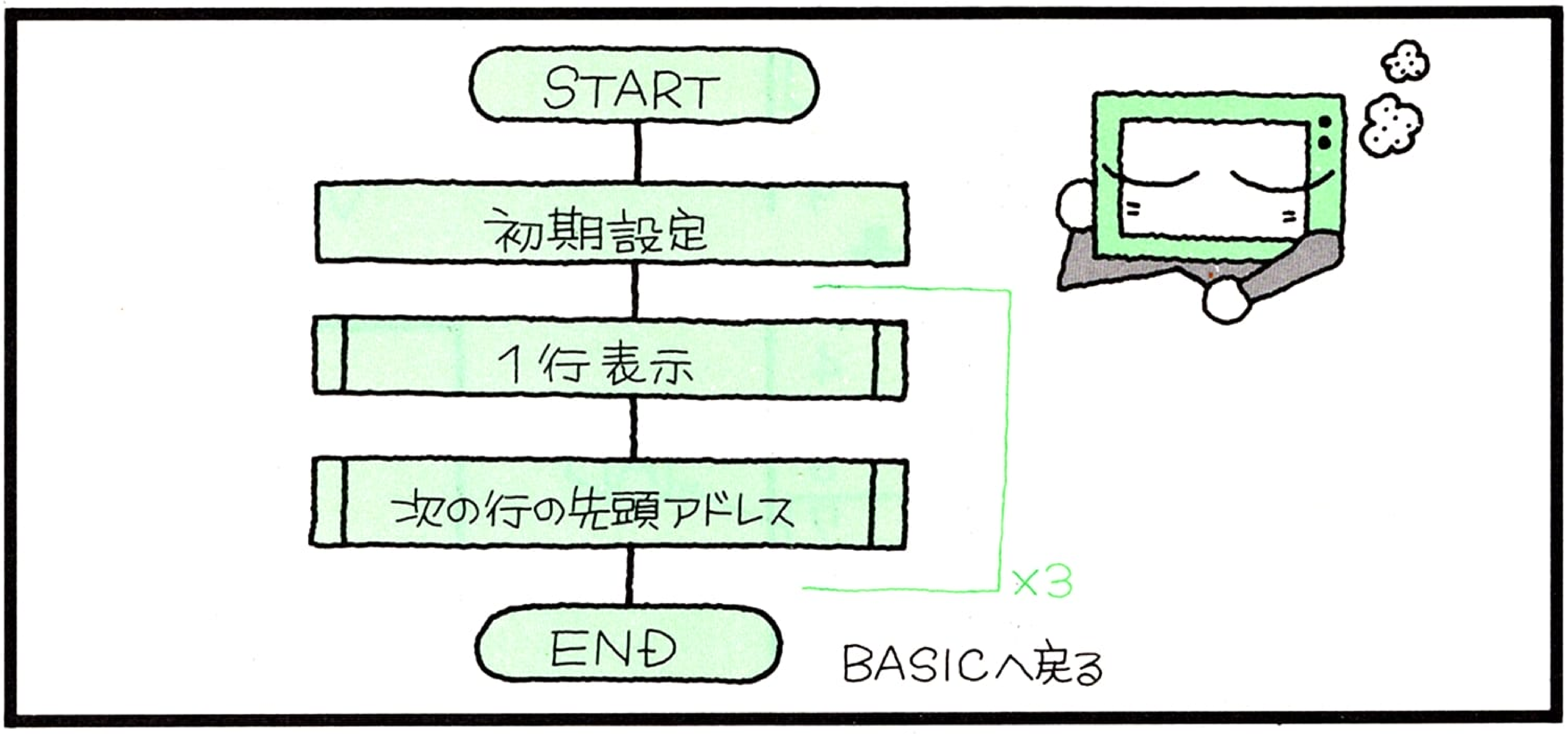


レジスタはこう使います。

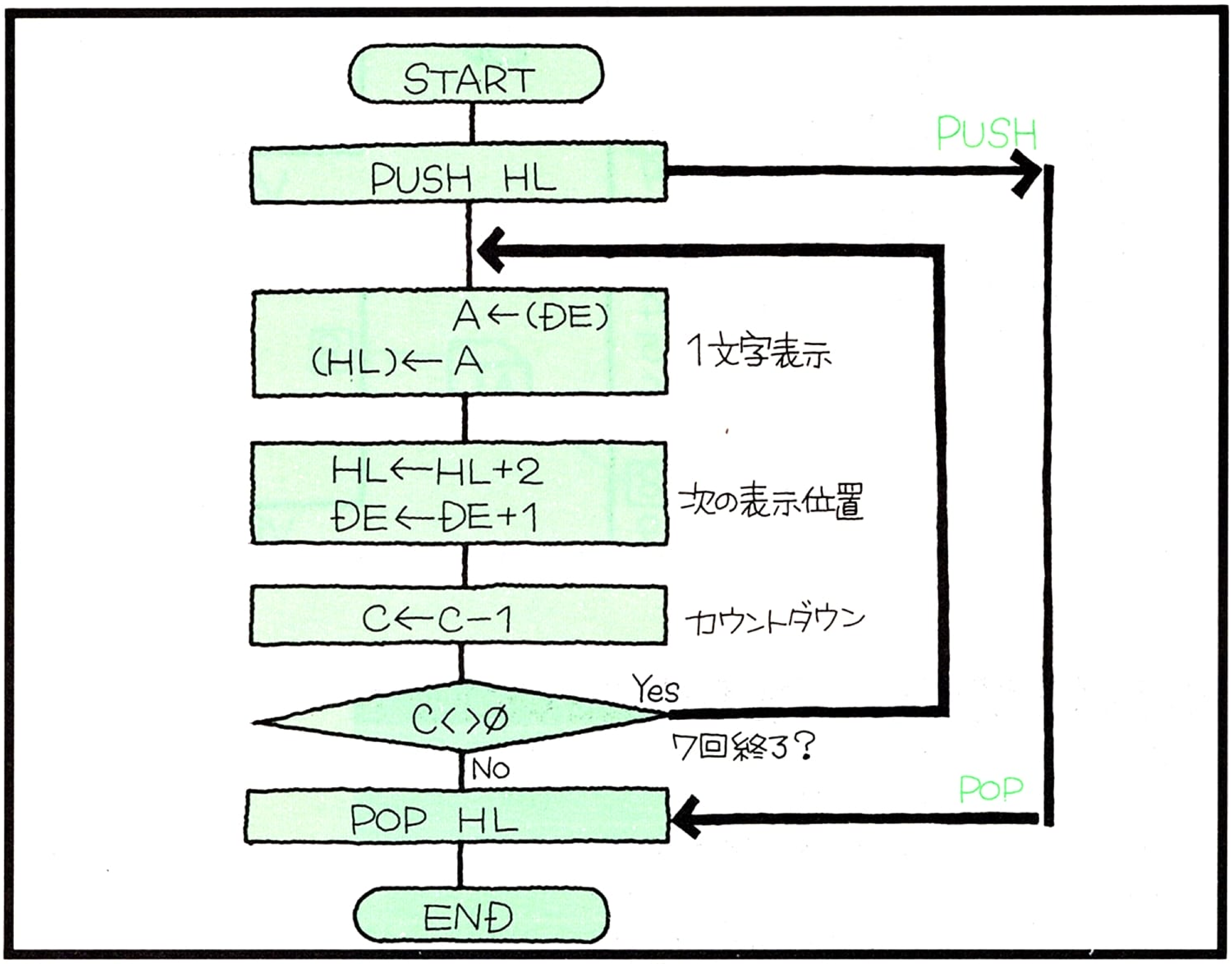
レジスタ	役　割	
B	縦用カウンタ	加算用
C	横用カウンタ	(120)
HL	表示アドレス　(書き込み用)	
DE	データアドレス (読み出し用)	
A	メモリのデータをVRAMにコピーする	

フローチャート

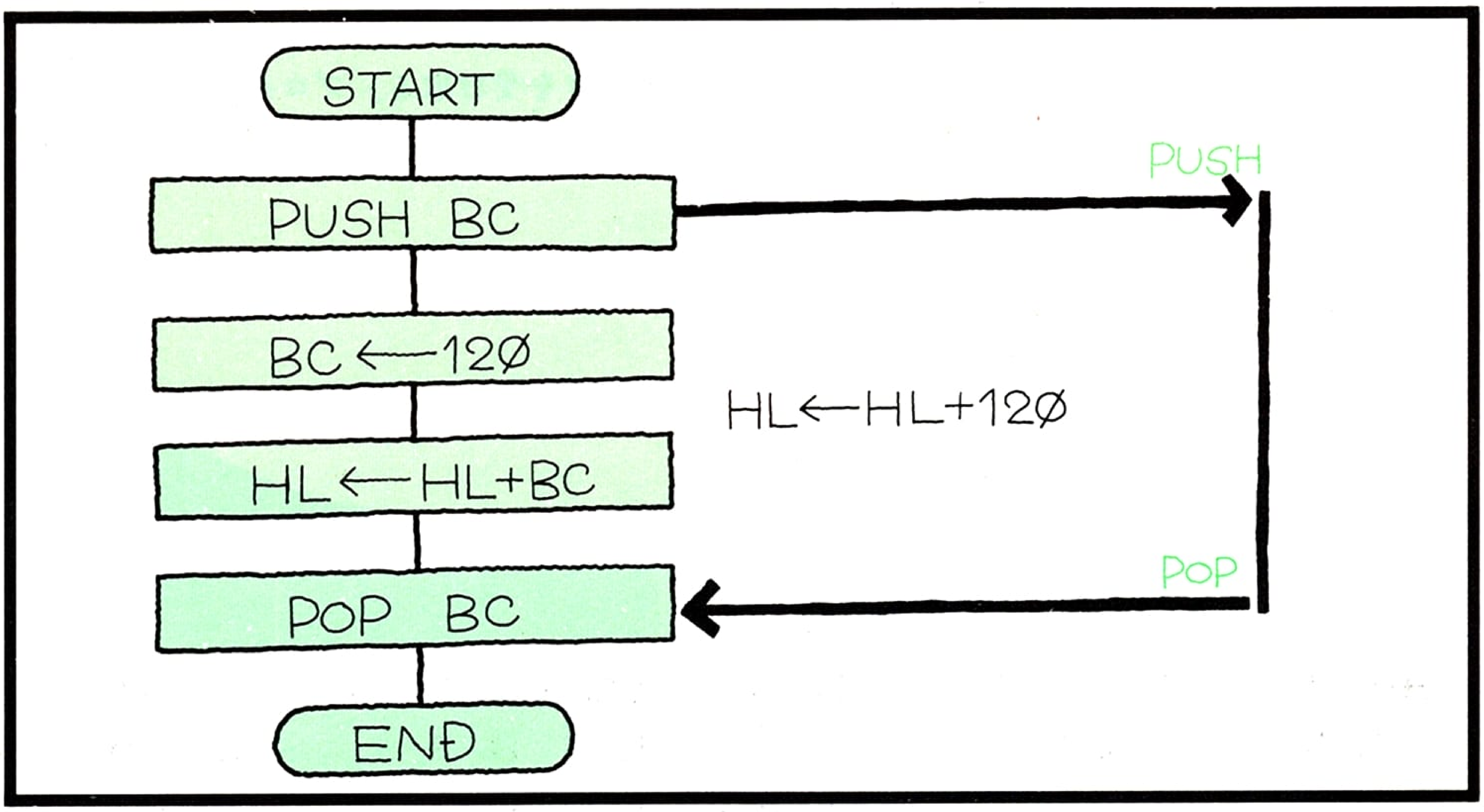
まず、メインループ。



そして、1行表示ルーチン。



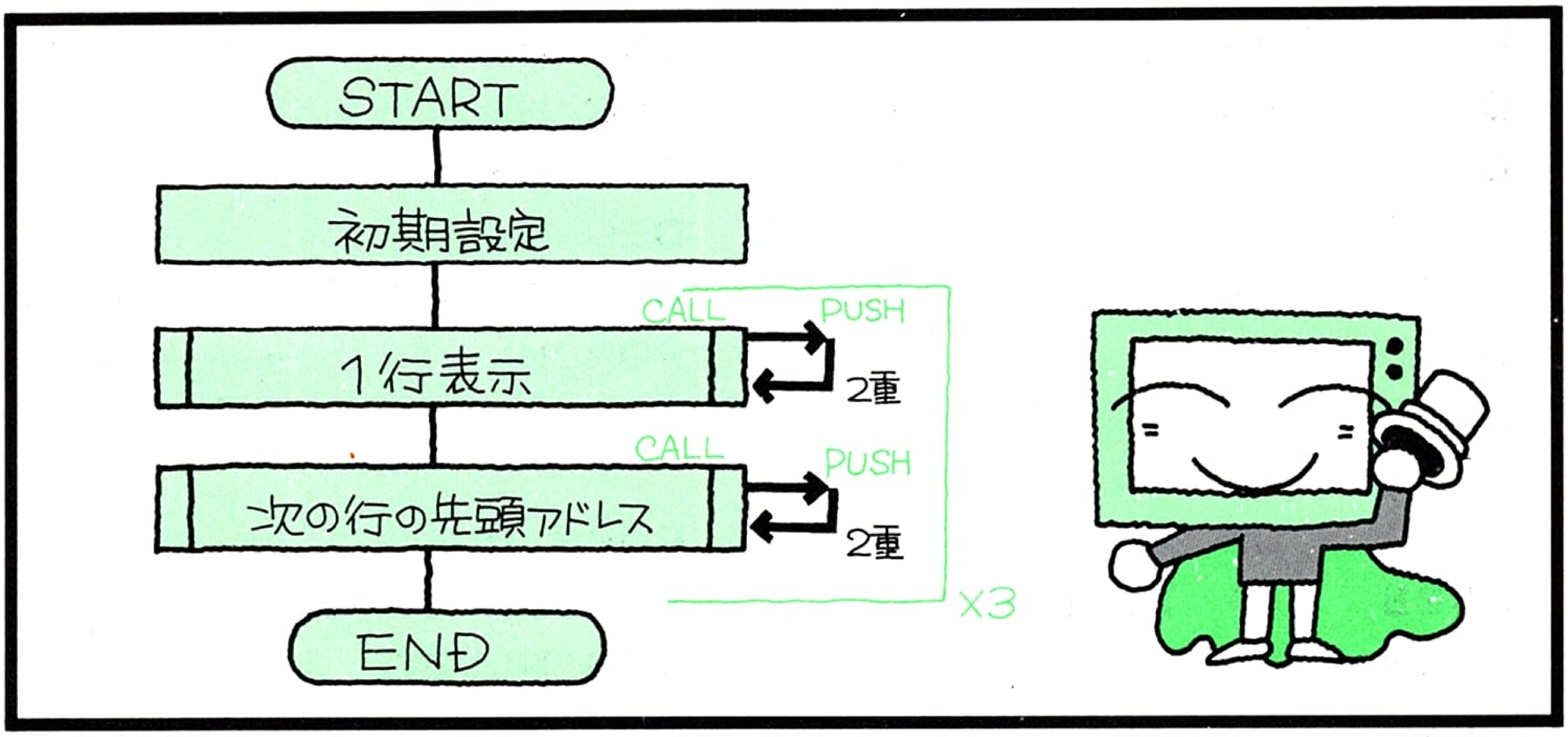
行の先頭アドレスの計算はこうします。



これであとは、プログラムにするだけです。
ただし、気をつけなければならないことが1つだけあります。
それは、

ＢＡＳＩＣから呼ぶマシン語サブルーチンの中には、ＰＵＳＨやＣＡＬＬをあわせて７重までしか使えない。

ということです。
今回は、



ですから、最高でも２重だったので大丈夫です。
これからは、ＣＡＬＬやＰＵＳＨを多く使う時注意しましょう。

プログラム

先程の「ＢＡＳＩＣからデータを受け取る」プログラムがＢ００８で終っていたので、続きに書いていくこのプログラムは「Ｂ００９」から始めていきましょう。

B009	11 00 C0	MAIN	LD DE, C000	DE：DATA
B00C	06 03		LD B, 3	B：Yカウンタ
B00E	0E 07	LOOPY	LD C, 7	
B010	CD 1B B0		CALL LINE1	1行表示！
B013	CD 27 B0		CALL AD120	先頭アドレス！
B016	05		DEC B	×3
B017	C2 0E B0		JP NZ, LOOPY	
B01A	C9		RET	RETURN
B01B	E5	LINE1	PUSH HL	
B01C	1A	LOOPX	LD A, (DE)	
B01D	77		LD (HL), A	
B01E	23		INC HL	1文字表示
B01F	23		INC HL	
B020	13		INC DE	(40字モード)
B021	0D		DEC C	
B022	C2 1C B0		JP NZ, LOOPX	
B025	E1		POP HL	
B026	C9		RET	
B027	C5	AD120	PUSH BC	
B028	01 78 00		LD BC, 0078	HL←HL＋120
B02B	09		ADD HL, BC	
B02C	C1		POP BC	
B02D	C9		RET	

実行！

では、プログラムを入力します。

マシン語は**プログラム**と、

```
B000 5E 23 56 EB 24 2C CD 9D
B008 42 11 00 C0 06 03 0E 07
B010 CD 1B B0 CD 27 B0 05 C2
B018 0E B0 C9 E5 1A 77 23 23
B020 13 0D C2 1C B0 E1 C9 C5
B028 01 78 00 09 C1 C9
```

データ

```
C000 00 E8 E8 E8 E8 00 00 84
C008 87 87 87 87 56 56 00 87
C010 00 00 87 94 94
```

ＢＡＳＩＣはこれです。

```
10 CLEAR ,&HAFFF
20 DEFINT A-Z
30 DEF USR=&HB000
40 INPUT "X=",X
50 INPUT "Y=",Y
60 A=USR(X*256+Y)
70 END
```

ステゴザウルスは自由に現われます。



5・3　プテラノドン

ブロントザウルス、ステゴザウルスとくればもう次はプテラノドンしかありません。

プテラノドンを出現させてみましょう。



今度は自由に飛び回れるようにしてやります。

キーボードから、その方向を入力して飛行させてやろうというのです。



以上の5パートにわけて作っていくことにします。

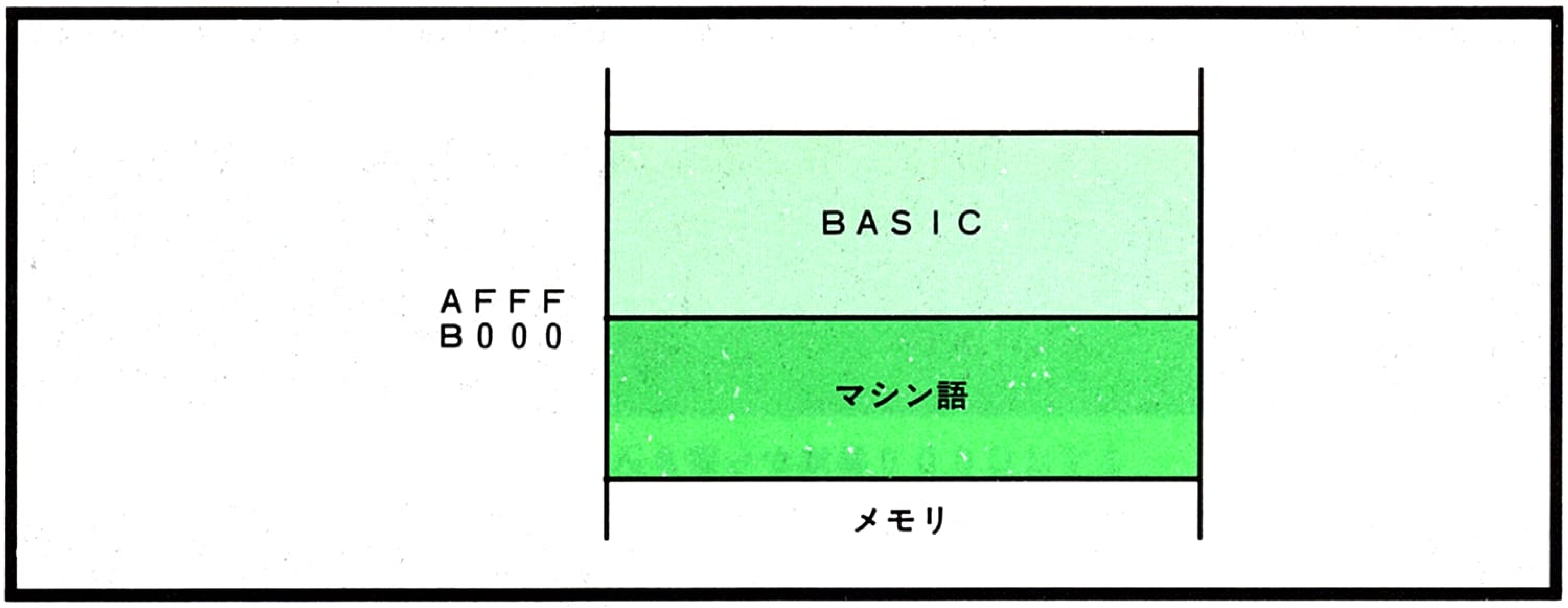
1　初期設定

プログラムを作る以前に考えておかなくてはならないのが、

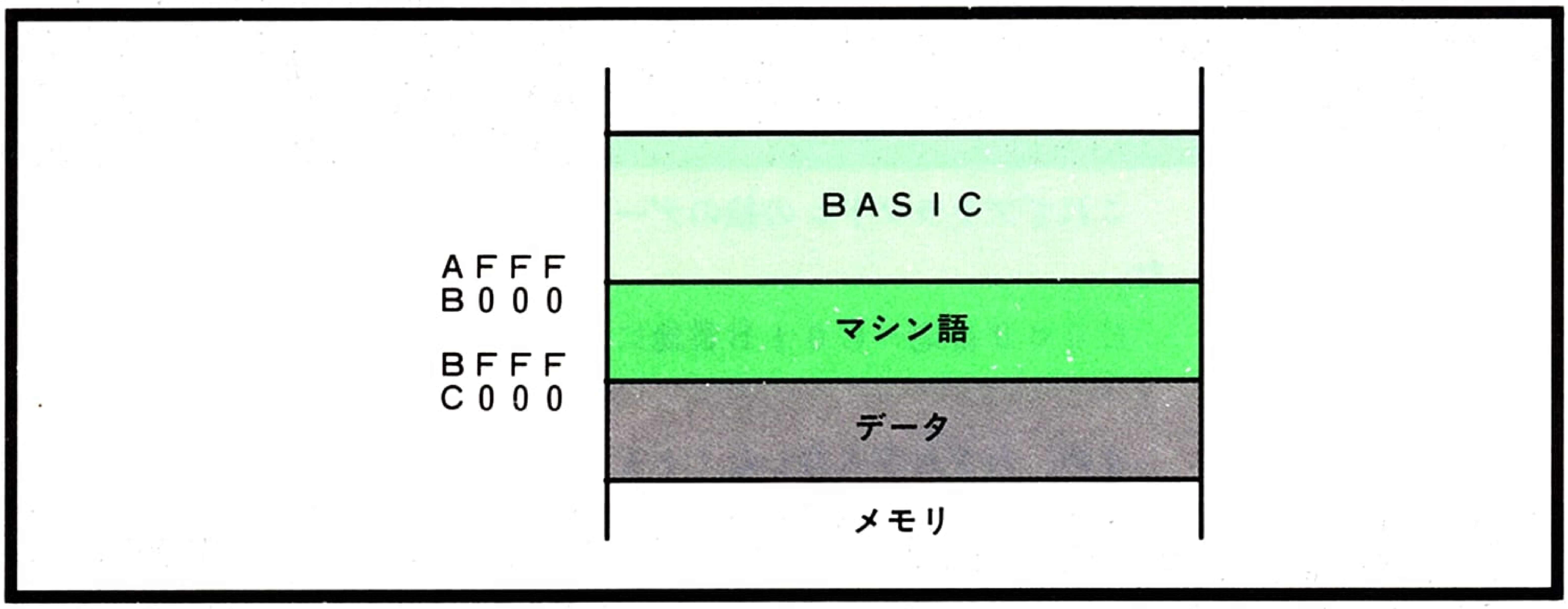
「メモリのどこを使うか」

です。

まず、**ＡＦＦＦ番地までをＢＡＳＩＣ、Ｂ０００番地以降をマシン語の領域**とします。



次に、**ＢＦＦＦ番地までをマシン語のプログラム、Ｃ０００以降をデータの領域**と考えるのです。



これで、使用するメモリの領域が決まりました。

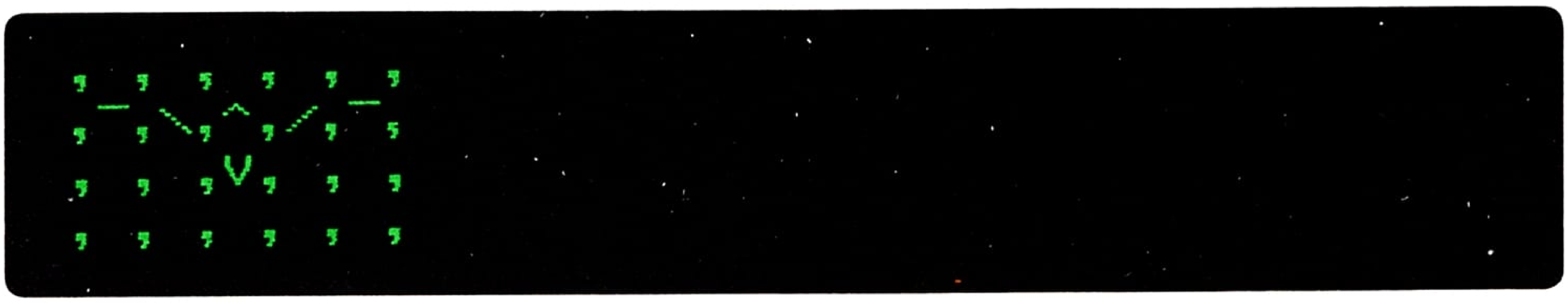
ところでこのデータの領域には一体どんなデータが入るのでしょう。

もちろん、プテラノドンの絵のデータですが、その他にもプテラノドンを表示するＸＹ座標のデータも必要です。

なぜなら、ＢＡＳＩＣからマシン語に送る情報は移動方向のデータだけですから、プテラノドンの表示データはマシン語のプログラムの中に用意しておかなくてはならないのです。

そこでメモリにデータを書き込んでおこうというわけです。

プテラノドンが上下左右に動き回る時、足あとを残すような格好の悪い事をしないためにもまわりをスペースで囲んでスペースごと動かすようにしましょう。



これを**アスキーコードのデータに変換**して

```
20,20,20,20,20,20,20
20,94,EF,5E,EE 94,20
20,20,20,56,20,20,20
20,20,20,20,20,20,20
```

まずは**Ｃ０００番地から書き込む**のです。

```
C000 20 20 20 20 20 20 20 20
C008 94 EF 5E EE 94 20 20 20
C010 20 56 20 20 20 20 20 20
C018 20 20 20 20 00 FF 00 FF
C020 00 FF 00 FF 00 FF 00 FF
C028 00 FF 00 FF 00 FF 00 FF
```

これでプテラノドンの絵のデータがどこに入っているかわかりました。

Ｃ０００番地～Ｃ０１Ｂ番地に入っているのですね。

まあ、わざわざ入力しなくとも、

```
print hex$(&hc000+7*4-1)
C01B
Ok
█
```

という計算によって求めることもできたわけですが……。

ともあれ、Ｃ０００～Ｃ０１Ｂのメモリは絵のデータですから、それ以降のメモリにＸＹ座標のデータを書き込むとします。

わかりやすいように、**Ｃ０２０をＸ座標、Ｃ０２１をＹ座標のデータ格納場所**としましょう。

2　方向を入力する

これだけ決めておけば、ＢＡＳＩＣで楽にプログラムを組むことができます。

要は、初期設定とキーボードから受けとった情報をマシン語に送ればそれでいいというわけですね。

マシン語には上下方向と左右方向の２つの情報を送らねばなりません。そのため、使える数値は０～255です。よってマイナス方向へ移動するということをそのまま伝えることができません。

そこで、

０　：　移動しない
１　：　＋１移動する
２　：　－１移動する

という約束をしておいてマシン語に送り、受け取ったマシン語の方でこの処理をするようにする方法をとります。

[2] [4] [6] [8] のキーを押すことによって変数ＶＸ、ＶＹに情報が入るようにしてみます。

```
100 I0=INP(0) :I1=INP(1)
110 VX=0 :VY=0
120 IF (I1 AND 1 )=0 THEN VY= 2 :' -1
130 IF (I0 AND 4 )=0 THEN VY= 1 :' +1
140 IF (I0 AND 16)=0 THEN VX= 2 :' -1
150 IF (I0 AND 64)=0 THEN VX= 1 :' +1
160 LOCATE 0,0:PRINT "VX","VY"
170 LOCATE 0,1:PRINT  VX , VY
180 GOTO 100
```

ＲＵＮさせると、

```
VX           VY
 2            1
```

ＶＸ、ＶＹに確かに情報が入ることがわかります。

あとは、**これをＵＳＲ関数を使ってマシン語に送ればいい**のです。

ＢＡＳＩＣ部

まず､｢ＣＬＥＡＲ｣でＢＡＳＩＣ領域の終わりをＡＦＦＦ番地と決め、画面をカラーの40字モードにして、マシン語の呼び出し番地をＢ０００と宣言します。

あとは、プテラノドンの最初の位置を、たとえば横10、縦12と入力して、キーボードからの方向入力に備えます。

```
100 REM ===== ｼｮｷｾｯﾃｲ =====
110 CLEAR ,&HAFFF
120 DEFINT A-Z
130 WIDTH 40,25 :CONSOLE 0,25,0,1
140 COLOR 5      :CLS
150 DEF USR=&HB000
160 POKE &HC020,10 :' X data (10)
170 POKE &HC021,12 :' Y data (12)
180 REM ===== inkey$? =====
190 I0=INP(0) :I1=INP(1)
200 VX=0 :VY=0
210 IF (I1 AND 1 )=0 THEN VY= 2 :' -1
220 IF (I0 AND 4 )=0 THEN VY= 1 :' +1
230 IF (I0 AND 16)=0 THEN VX= 2 :' -1
240 IF (I0 AND 64)=0 THEN VX= 1 :' +1
250 '
260 A=USR(VX*256+VY) :' ﾏｼﾝｺﾞ ﾍ
270 '
280 GOTO 190
```

これでＢＡＳＩＣ部は完成です。

と言っても､｢ＲＵＮ｣は待って下さい。まだ、マシン語プログラムがないので暴走してしまいます。

それでもかまわないのなら無理に止めたりはいたしませんが、お楽しみは後にとっておいて、マシン語プログラムが出来上ってからＲＵＮさせる方がきっといいと思うのです。

3　データを受けとる

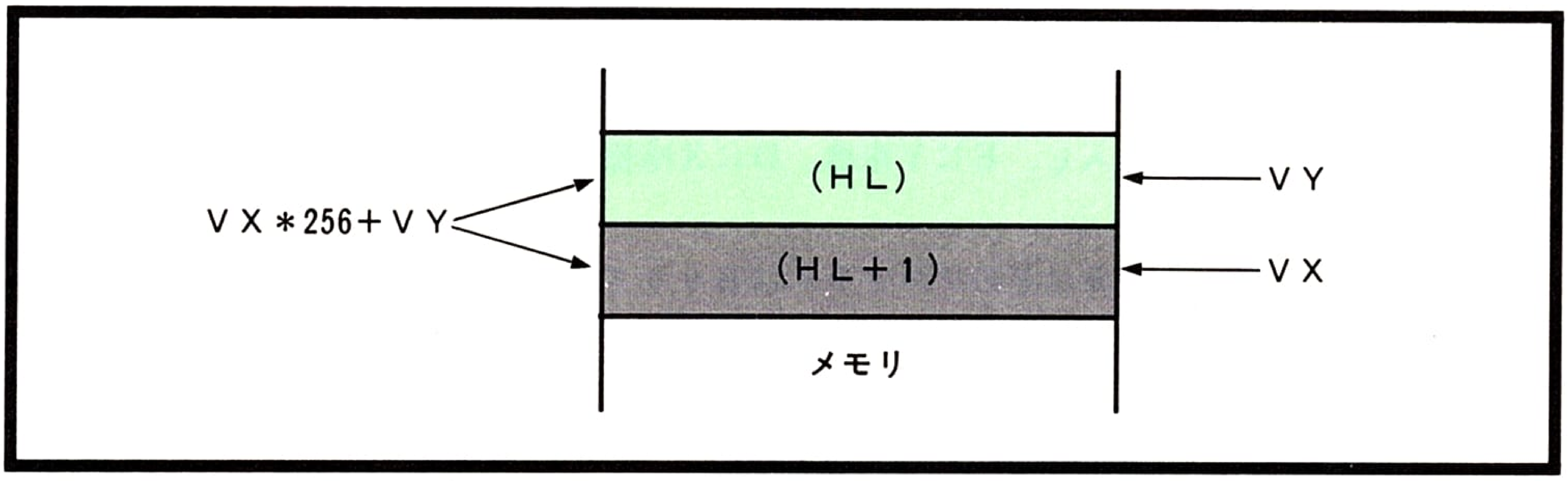
マシン語のプログラムです。

まず最初は、ＢＡＳＩＣから送られて来た**「プテラノドンを動かす方向のデータ」**を読み取ります。

これをしないことには、アンテナのとれたラジコンのようにどこに飛んで行くかわからないからです。

でも、このアンテナは前に一度ステゴザウルスの時作ったことがあるので簡単ですね。

ＨＬレジスタが指しているアドレスに下２ケタであるＶＹ成分が、その次のアドレスに上２ケタであるＶＸ成分が入っているのでした。



これをＬＤ命令によって読み出してやればいいわけです。

ＣレジスタにＶＹ成分、ＢレジスタにＶＸ成分を読み出してやりましょう。

ちなみにスタートアドレスはＢ０００です。

アドレス	マシン語	ラベル	ニーモニック	コメント
			ORG　B000	START
B000	4E	START	LD　C, (HL)	C←VY
B001	23		INC　HL	
B002	46		LD　B, (HL)	B←VX

4　座標計算を行う

さて問題は座標計算です。今度のはＸＹ座標ではなく方向ベクトルだけしか与えられていないからです。

でもあわてることはありません。ＸＹ座標が与えられていないのなら**メモリからデータを読み出せばいい**のです。

そのあとで座標に運動する方向を加えて移動させればプテラノドンもちゃんと移動してくれます。

まずは、ＸＹ座標を読み出しましょう。ＢＣレジスタは方向ベクトルに、ＨＬレジスタはデータのアドレスを指すのに使用していますので、**ＤＥレジスタ**を使います。

ＨＬに、座標データの格納されているメモリのアドレスＣ０２０を代入し、ＥにＹ座標、ＤにＸ座標を読み出します。

先ほどのプログラムはＢ００２で終っていたので続きはＢ００３からです。

B003	21 20 C0		LD HL,0C020H	HL←データアドレス
B006	5E		LD E,(HL)	E←Y
B007	23		INC HL	
B008	56		LD D,(HL)	D←X

〈注〉上のリストの「０Ｃ０２０Ｈ」の「Ｈ」は、「Ｃ０２０」が16進数であることを意味します。

これができればあとは、ＢＣレジスタに入っている情報によって、ＤＥの中の座標を足したり引いたりします。

ＢＣの値が「１」なら「＋１」、「２」なら「－１」を行うのでしたね。

ＢＡＳＩＣで例えるとこうなるでしょうか。

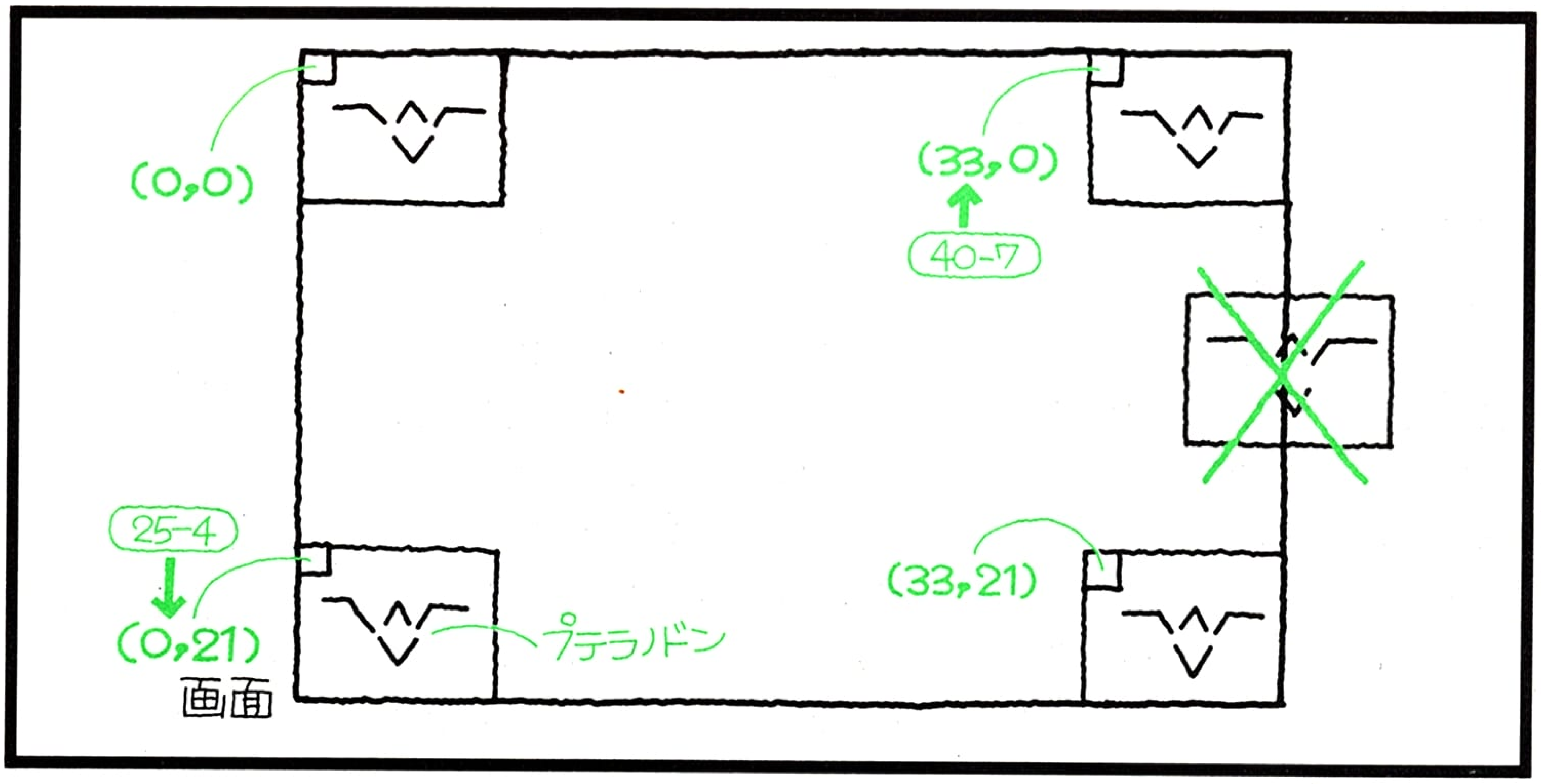
```
100 IF B=1 THEN D=D+1
110 IF B=2 THEN D=D-1
120 IF C=1 THEN E=E+1
130 IF C=2 THEN E=E-1
```

ただし、マシン語の条件判断の後は「ＧＯＴＯ」か「ＧＯＳＵＢ」しか使えないため少々面倒ですがこう書き変えることにします。

```
100 IF B=1 THEN GOSUB 200
110 IF B=2 THEN GOSUB 220
120 IF C=1 THEN GOSUB 240
130 IF C=2 THEN GOSUB 260
140 END
200 D=D+1
210 RETURN
220 D=D-1
230 RETURN
240 E=E+1
250 RETURN
260 E=E-1
270 RETURN
```

ところでプテラノドンは７×４の絵でできています。そのためあまり画面の端に行くとはみ出してしまうのです。

そこで、もし画面の端に来たら演算をもう行わない条件判断をつけ加えることにします。



ＢＡＳＩＣで表記してみましょう。

```
100 IF B=1 THEN GOSUB 200
110 IF B=2 THEN GOSUB 220
120 IF C=1 THEN GOSUB 240
130 IF C=2 THEN GOSUB 260
140 END
200 IF D=33 THEN 210
205 D=D+1
210 RETURN
220 IF D=0  THEN 230
225 D=D-1
230 RETURN
240 IF E=21 THEN 250
245 E=E+1
250 RETURN
260 IF E=0  THEN 270
265 E=E-1
270 RETURN
```

少し長めのリストになってしまいました。

でもやってることは単純なことのくり返し４回にすぎないので、とても簡単なはずです。

ではマシン語に訳してみましょう。続きはＢ００９からです。

ただし、140行で終わってしまってはいけないのでＪＵＭＰして抜け出すようにしておきます。

B009	05	XPLUS	DEC B	100行
B00A	CC 1C B0		CALL Z, XP	
B00D	05	XMINS	DEC B	110行
B00E	CC 24 B0		CALL Z, XM	
B011	0D	YPLUS	DEC C	120行
B012	CC 2C B0		DEC C	
B015	0D	YMINS	CALL Z, YP	130行
B016	CC 34 B0		CALL Z, YM	
B019	C3 3C B0	END	JUMP NEXT	140行

B01C	7A	XP	LD　A, D	
B01D	FE 21		CP　A, 33	200行
B01F	CA 23 B0		JP　Z, RETXP	
B022	14		INC D	205行
B023	C9	RETXP	RET	210行
B024	7A	XM	LD　A, D	
B025	FE 00		CP　A, 0	220行
B027	CA 2B B0		JP　Z, RFTXM	
B02A	15		DEC D	225行
B02B	C9	RETXM	RET	230行
B02C	7B	YP	LD　A, E	
B02D	FE 15		CP　A, 21	240行
B02F	CA 33 B0		JP　Z, RETYP	
B032	1C		INC E	245行
B033	C9	RETYP	RET	250行
B034	7B	YM	LD　A, E	
B035	FE 00		CP　A, 0	260行
B037	CA 3B B0		JP　Z, RETYM	
B03A	1D		DEC E	265行
B03B	C9	RETYM	RET	270行
B03C		NEXT		

これでＢＡＳＩＣからの情報によってプテラノドンの位置移動の計算ができました。

ただし、これはＸＹ座標による計算でしかないわけですからＶＲＡＭアドレスに変換する必要がありますし、それよりもまず移動計算の終わったＤＥのデータをメモリに書きとめておく必要があります。

Ｃ０２０にＤの値、Ｃ０２１にＥの値を書き込んでおかないと、次にこのプログラムを呼び出してもプテラノドンは同じ位置から動いていないわけですから自由に飛びまわることができないということになります。

読み出した時の逆をします。

B03C	21 20 C0	NEXT	LD HL,0C020H	HL←データアドレス
B03F	73		LD (HL), E	Y←E
B040	23		INC HL	
B041	72		LD (HL), D	X←D

これでＤＥのデータは保存されました。

次に**ＶＲＡＭアドレス**を求めます。

Ｈに X 座標＋１、ＬにＹ座標＋１を代入してから「４２９Ｄ」をＣＡＬＬするとＨＬにＶＲＡＭアドレスが代入されました。

と言うことは、まずＨＬにＸＹ座標が入っていなければ話になりません。**ＤＥとＨＬを交換**しておく必要があるのです。

それでは、即、マシン語にしてみましょう。

B042	EB		EX DE, HL	DE↔HL
B043	24		INC H	「＋１」
B044	2C		INC L	
B045	CD 9D42		CALL 429DH	VRAM計算

5　プテラノドン表示

プテラノドンを表示するＶＲＡＭアドレスが求まったなら、次には何をするべきか。

もちろん、**プテラノドンの表示**です。

表示させるプログラムはブロントザウルス、ステゴザウルスとみっちりやりましたのでもう簡単ですね。

さっそく、メインルーチンを作ってみましょう。プログラムはＢ０４８から続けます。

B048	11 00 C0	MAIN	LD DE,0C000H	DE：DATA
B04B	06 04		LD B,4	B：Yカウンタ
B04D	0E 07	LOOPY	LD C,7	C：Xカウンタ
B04F	CD 5A B0		CALL LINE1	1行表示！
B052	CD 66 B0		CALL AD120	先頭アドレス
B055	05		DEC B	
B056	C2 4D B0		JP NZ,LOOPY	3回
B059	C9		RET	RETURN TO BASIC

あと、１行表示と先頭アドレスを求めるサブルーチンを作ればもうそれで完成です。

Ｂ０５Ａから続けます。

B05A	E5	LINE1	PUSH HL	
B05B	1A	LOOPX	LD A, (DE)	
B05C	77		LD (HL), A	1文字表示
B05D	23		INC HL	
B05E	23		INC HL	(40字モード)
B05F	13		INC DE	
B060	0D		DEC C	
B061	C2 5B B0		JP NZ, LOOPX	
B064	E1		POP HL	
B065	C9		RET	RETURN
B066	C5	AD120	PUSH BC	
B067	01 78 00		LD BC, 120	HL←HL+120
B06A	09		ADD HL, BC	
B06B	C1		POP BC	
B06C	C9		RET	RETURN

データがこれで、

```
C000 20 20 20 20 20 20 20 20
C008 94 EF 5E EE 94 20 20 20   ‾\^/‾
C010 20 56 20 20 20 20 20 20    V
C018 20 20 20 20
```

マシン語がこれです。

```
B000 4E 23 46 21 20 C0 5E 23   N#F! ﾀ^#
B008 56 05 CC 1C B0 05 CC 24   V ﾌ ｰ ﾌ$
B010 B0 0D CC 2C B0 0D CC 34   ｰ ﾌ,ｰ ﾌ4
B018 B0 C3 3C B0 7A FE 21 CA   ｰﾃ<ｰz▒!ﾊ
B020 23 B0 14 C9 7A FE 00 CA   #ｰ ﾉz▒ ﾊ
B028 2B B0 15 C9 7B FE 15 CA   +ｰ ﾉ{▒ ﾊ
B030 33 B0 1C C9 7B FE 00 CA   3ｰ ﾉ{▒ ﾊ
B038 3B B0 1D C9 21 20 C0 73   ;ｰ ﾉ! ﾀs
B040 23 72 EB 24 2C CD 9D 42   #r♠$,ﾍ､B
B048 11 00 C0 06 04 0E 07 CD     ﾀ    ﾍ
B050 5A B0 CD 66 B0 05 C2 4D   Zｰﾍfｰ ﾂM
B058 B0 C9 E5 1A 77 23 23 13   ｰﾉ◣ w##
B060 0D C2 5B B0 E1 C9 C5 01    ﾂ[ｰ ﾋﾉﾅ
B068 78 00 09 C1 C9            x  ﾁﾉ
```

RUNしましょう。

プテラノドンの出現です。



★ SPECIAL LIST ★

プログラム自体は先ほどので、一応完成です。

しかし、バックがまっ黒ではあまりにもさびしいものですから、プテラノドンが飛びまわったであろう中生代の海を描いてみました。

背景を描くだけでかなりリアルになるものです。



ここにはＢＡＳＩＣのリストしか載せてありませんが、マシン語のリストは先ほどのと同じですのでそれを利用して下さい。

PROGRAM LIST

```
10 REM ┌──────────────────────────────┐
20 REM │                              │
30 REM │  <<<   ﾌﾟﾃﾗﾉﾄﾞﾝ    >>>       │
40 REM │                              │
50 REM │                              └
60 REM │   Y.MORI      1984.6.1       ┌
70 REM │                              │
80 REM └──────────────────────────────┘
90 '
100 REM =====  ｼｮｷｾｯﾃｲ  =====
110 CLEAR ,&HAFFF
120 DEFINT A-Z
130 WIDTH 40,25 :CONSOLE 0,25,0,1
140 GOSUB 300 :' SCREEN
150 DEF USR=&HB000
160 POKE &HC020,10 :' X data (10)
170 POKE &HC021,12 :' Y data (12)
180 REM ===== inkey$? =====
190 I0=INP(0) :I1=INP(1)
200 VX=0 :VY=0
210 IF (I1 AND 1 )=0 THEN VY=2 :' -1
220 IF (I0 AND 4 )=0 THEN VY=1 :' +1
230 IF (I0 AND 16)=0 THEN VX=2 :' -1
240 IF (I0 AND 64)=0 THEN VX=1 :' +1
250 '
260 A=USR(VX*256+VY) :' ﾏｼﾝｺﾞ ﾍ
270 '
280 GOTO 190
290 '
300 REM ===== S T A G E =====
310 COLOR 7 :CLS 3
320 FOR I=1 TO 293
330  READ X,Y,Z
340  IF Z=1 THEN LINE(X,Y)-(X,Y),1
350  IF Z=0 THEN LINE     -(X,Y),1
360 NEXT
370 T1$=CHR$(&H55)+CHR$(&HFF)+CHR$(&HFF)
380 T2$=CHR$(0)+CHR$(&H55)+CHR$(&HAA)
```

画面 (130–140)
マシン語初期化 (150–170)
メインループ (190–280)
輪郭 (320–360)
地面 (370)
山 (380)

```
390 T3$=CHR$(0)+CHR$(&H55)+CHR$(0)         ← 岩（暗）
400 T4$=CHR$(0)+CHR$(&HBB)+CHR$(&H11)+CH   ← 岩（明）
R$(0)+CHR$(&HEE)+CHR$(&H44)
410 T5$=CHR$(&HFF)+CHR$(0)+CHR$(&H55)      ← 空
420 FOR I=1 TO 9            ┐
430  READ X,Y,C             ├─ 色
440  PAINT(X,Y),C,1         ┘
450 NEXT
460 PAINT(600,199),T1$,1    ┐
470 PAINT(200,110),T2$,1    │
480 PAINT(300,110),T2$,1    │
490 PAINT(620,110),T2$,1    │
500 PAINT(240,120),T3$,1    │
510 PAINT(350,120),T3$,1    ├─ 中間色
520 PAINT(225,115),T4$,1    │
530 PAINT(350,110),T4$,1    │
540 PAINT(460,100),T4$,1    │
550 PAINT(  0,  0),T5$,1    ┘
560 RESTORE
570 FOR I=1 TO 293
580  READ X,Y,Z                          ┐
590  IF Z=1 THEN LINE(X,Y)-(X,Y),0       ├─ 輪郭（黒）
600  IF Z=0 THEN LINE      -(X,Y),0      ┘
610 NEXT
620 RETURN
630 '
640 REM ===== D A T A S =====
650 '                          以下はＬＩＮＥ文データ
660 DATA 51, 199, 1, 71, 197, 0, 119, 19
3, 0, 141, 192, 0, 173, 188, 0, 186, 188
, 0, 222, 185, 0, 258, 183, 0, 277, 182,
 0, 309, 179, 0
670 DATA 327, 179, 0, 343, 176, 0, 357,
174, 0, 369, 173, 0, 394, 169, 0, 429, 1
64, 0, 456, 161, 0, 475, 160, 0, 509, 15
4, 0, 546, 149, 0
680 DATA 580, 145, 0, 600, 141, 0, 617,
139, 0, 638, 137, 0, 639, 137, 0, 600, 1
41, 1, 573, 142, 0, 568, 144, 0, 544, 14
5, 0, 513, 149, 0
```

```
690 DATA 482, 153, 0, 468, 156, 0, 454,
156, 0, 429, 161, 0, 386, 166, 0, 321, 1
71, 0, 243, 175, 0, 147, 183, 0, 123, 18
4, 0, 104, 188, 0
700 DATA 86, 188, 0, 77, 187, 0, 68, 189
, 0, 45, 194, 0, 21, 194, 0, 0, 197, 0,
0, 189, 1, 17, 186, 0, 48, 183, 0, 95, 1
82, 0
710 DATA 128, 179, 0, 188, 173, 0, 211,
173, 0, 259, 167, 0, 334, 164, 0, 392, 1
59, 0, 438, 155, 0, 459, 153, 0, 478, 15
1, 0, 505, 150, 0
720 DATA 473, 151, 1, 426, 153, 0, 309,
160, 0, 257, 161, 0, 217, 165, 0, 185, 1
67, 0, 167, 169, 0, 138, 170, 0, 106, 17
4, 0, 67, 176, 0
730 DATA 29, 181, 0, 14, 180, 0, 0, 182,
 0, 0, 172, 1, 28, 170, 0, 51, 171, 0, 9
9, 168, 0, 177, 161, 0, 234, 160, 0, 244
, 159, 0
740 DATA 253, 155, 0, 261, 154, 0, 284,
155, 0, 298, 155, 0, 318, 154, 0, 338, 1
52, 0, 383, 153, 0, 383, 153, 0, 424, 15
1, 0, 453, 149, 0
750 DATA 414, 151, 1, 370, 150, 0, 327,
149, 0, 286, 152, 0, 254, 151, 0, 246, 1
53, 0, 238, 155, 0, 222, 156, 0, 200, 15
6, 0, 158, 159, 0
760 DATA 72, 165, 0, 27, 164, 0, 0, 166,
 0, 23, 161, 1, 33, 161, 0, 54, 159, 0,
82, 158, 0, 100, 159, 0, 124, 156, 0, 15
3, 157, 0
770 DATA 133, 156, 1, 89, 154, 0, 56, 15
7, 0, 39, 160, 0, 492, 140, 1, 549, 138,
 0, 574, 137, 0, 541, 136, 0, 509, 138,
0, 497, 140, 0
780 DATA 0, 121, 1, 210, 121, 0, 267, 12
0, 1, 333, 120, 0, 533, 118, 1, 545, 118
, 0, 598, 118, 1, 639, 118, 0, 639, 121,
 1, 614, 122, 0
```

```
790 DATA 557, 124, 0, 493, 127, 0, 449,
129, 0, 387, 129, 0, 364, 131, 0, 315, 1
30, 0, 328, 125, 0, 334, 120, 0, 340, 11
7, 0, 352, 104, 0
800 DATA 361, 100, 0, 374, 96, 0, 382, 8
6, 0, 388, 82, 0, 391, 81, 0, 392, 85, 0
, 386, 88, 0, 381, 96, 0, 393, 90, 0, 38
8, 98, 0
810 DATA 378, 105, 0, 379, 100, 0, 377,
99, 0, 374, 103, 0, 375, 107, 0, 371, 10
9, 0, 359, 111, 0, 347, 115, 0, 336, 120
, 0, 391, 81, 1
820 DATA 399, 80, 0, 405, 83, 0, 409, 85
, 0, 413, 86, 0, 421, 84, 0, 436, 82, 0,
 439, 83, 0, 443, 90, 0, 449, 96, 0, 460
, 103, 0
830 DATA 464, 103, 0, 473, 108, 0, 477,
114, 0, 472, 107, 1, 472, 100, 0, 474, 9
7, 0, 477, 90, 0, 477, 85, 0, 468, 95, 0
, 456, 100, 0
840 DATA 477, 85, 1, 480, 81, 0, 484, 79
, 0, 487, 74, 0, 491, 73, 0, 497, 77, 0,
 498, 84, 0, 506, 91, 0, 514, 99, 0, 515
, 103, 0
850 DATA 517, 99, 0, 521, 100, 0, 522, 1
04, 0, 529, 110, 0, 530, 115, 0, 534, 11
9, 0, 536, 122, 0, 542, 121, 0, 548, 116
, 0, 555, 112, 0
860 DATA 562, 111, 0, 569, 113, 0, 572,
115, 0, 574, 118, 0, 579, 116, 0, 583, 1
14, 0, 587, 117, 0, 595, 118, 0, 599, 11
8, 0, 620, 121, 0
870 DATA 401, 106, 1, 406, 106, 0, 415,
108, 0, 423, 111, 0, 426, 115, 0, 430, 1
18, 0, 430, 119, 0, 434, 124, 0, 287, 12
9, 1, 231, 129, 0
880 DATA 229, 130, 0, 208, 130, 0, 208,
128, 0, 212, 120, 0, 220, 117, 0, 228, 1
20, 0, 230, 124, 0, 233, 119, 0, 236, 11
6, 0, 235, 110, 0
```

```
890 DATA 238, 107, 0, 237, 104, 0, 235,
104, 0, 229, 107, 0, 222, 114, 0, 222, 1
18, 0, 237, 104, 1, 241, 106, 0, 241, 10
9, 0, 243, 110, 0
900 DATA 244, 110, 1, 250, 109, 0, 256,
109, 0, 257, 110, 0, 258, 115, 0, 263, 1
20, 0, 269, 121, 0, 273, 122, 0, 281, 12
5, 0, 285, 127, 0
910 DATA 287, 129, 0, 116, 121, 1, 130,
120, 0, 146, 118, 0, 184, 110, 0, 196, 1
05, 0, 202, 101, 0, 206, 101, 0, 209, 10
2, 0, 213, 104, 0
920 DATA 229, 107, 0, 255, 109, 1, 260,
106, 0, 264, 105, 0, 268, 105, 0, 272, 1
06, 0, 277, 108, 0, 283, 109, 0, 287, 10
9, 0, 297, 106, 0
930 DATA 312, 107, 0, 327, 109, 0, 340,
108, 0, 350, 106, 0, 528, 109, 1, 532, 1
09, 0, 534, 108, 0, 539, 108, 0, 543, 10
9, 0, 552, 107, 0
940 DATA 563, 105, 0, 565, 105, 0, 568,
106, 0, 571, 107, 0, 574, 105, 0, 583, 1
02, 0, 594, 100, 0, 601, 99, 0, 606, 97,
 0, 612, 93, 0
950 DATA 621, 89, 0, 629, 86, 0, 639, 81
, 0
960 DATA 0, 122, 1, 620, 120, 1, 540, 12
0, 1, 90, 155, 7, 540, 137, 7, 0, 170, 7
, 0, 185, 7, 100, 190, 7, 0, 190, 5
970 REM  ┌──────────────────────────────┐
980 REM  | FORESIGHT PLANNING SECTION  |
990 REM  └──────────────────────────────┘
```

第6章

ゲームの製作

6・1 ゲームの一例

ここに１つのゲームがあります。

〈注〉本章でのプログラムはディスクベースで動きます。ＰＣ－8801mkIIのディスク内蔵タイプをご使用の方は、ディップスイッチＳＷ－２の７番をＯＮにしてください。マシン語プログラムのディスクへのセーブ方法は198ページで解説します。

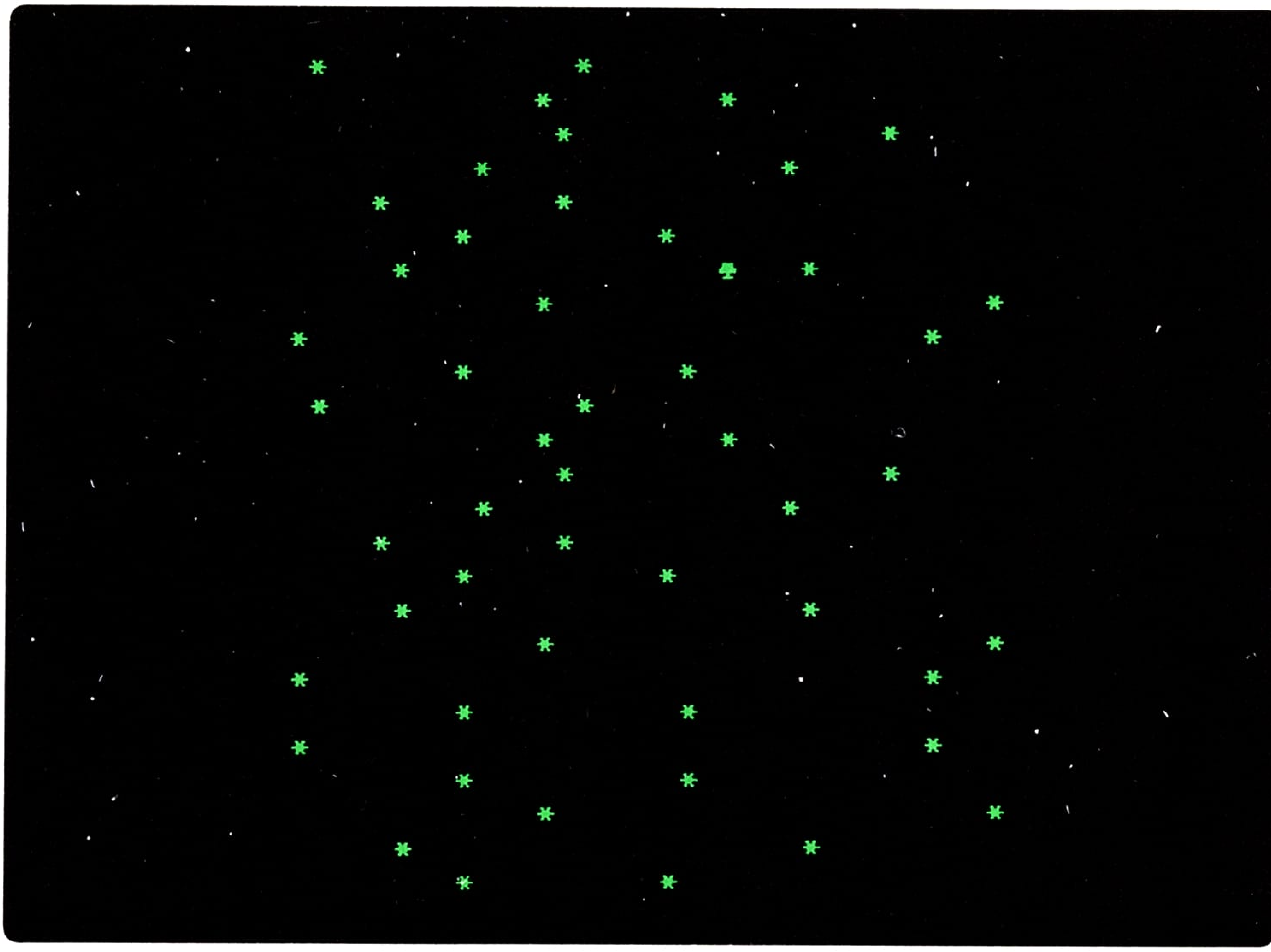
右端にいるクローバは、下から浮き上がってくるアスタリスク群をよけながら左端まで行かねばなりません。

左端まで行きつくとクローバは脱皮（？）をして少し下に降ります。今度は右端まで行かねばならないのです。

右端へ行きついたクローバは今度は左端へ……。

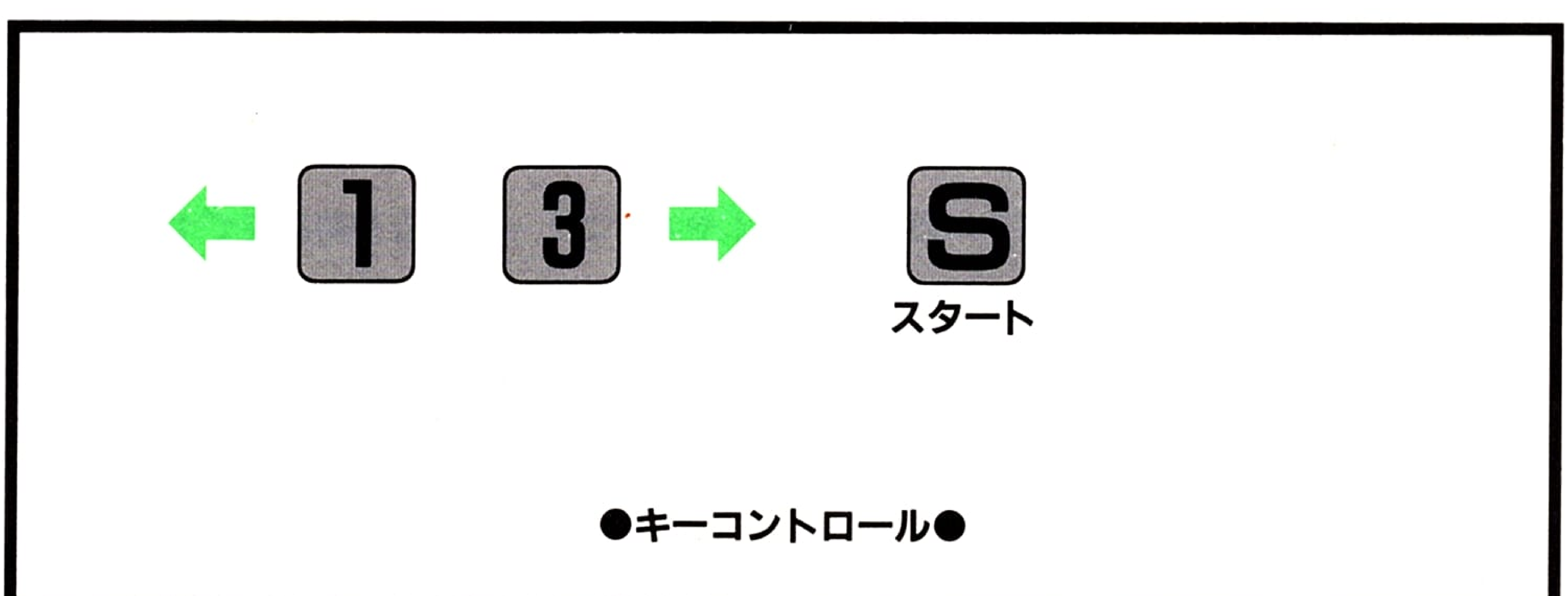
と、このような往復をくり返しながら下へ降りて行くというゲームです。

```
10 WIDTH 40,25:CONSOLE 0,25,0,1
20 DEFINT A-Z :COLOR 2 :CLS
30 X=37 :Y=6  :E=1
40 DEF FNP(X,Y)=PEEK(&HF3C8+X+X+Y*120)
50 I=INP(0)
60 V=(I=247)*(X<36)-(I=253)*(X>1)
70 X=X+V :P=FNP(X,Y)
80 IF P=42 THEN 210
90 IF X<>E THEN 110
100 Y=Y+2 :E=-35*(E=1)+1 :BEEP 1
110 COLOR 5 :LOCATE X-V,Y-1:PRINT " ";
120          LOCATE X  ,Y  :PRINT "♣";
130 FOR I=1 TO 2
140  LOCATE RND*34+2,24
150  COLOR 2:PRINT "*";
160 NEXT
170 PRINT
180 IF Y<22 THEN BEEP 0 :GOTO 50
190 COLOR 6
200 PRINT "Congratulation!":BEEP 0
210 LOCATE 13,10:PRINT (Y-6)*100;"Pts."
220 IF INP(4)<>247 THEN 220
230 RUN
```



一番下まで降りると「Congratulation!」というわけですが、これがなかなか難しい。

ですが、たったこれだけのゲームです。少し物足りないような気がしないでもありません。
そこで多少工夫をしてみることにしましょう。もちろんマシン語を使ってです。



●キーコントロール●

プログラムを読む

まず、このプログラム自体をはっきり理解しておくことにします。

というのは、ＢＡＳＩＣのメインプログラムでさえ理解していないうちからマシン語のプログラムを作ろうとしても失敗するのは目に見えているからです。

ＢＡＳＩＣならたとえ間違ったとしてもエラーが出て間違い箇所を指摘してくれます。

しかし、マシン語では間違ってもエラーが出ないはおろか間違い箇所を教えてもくれません。

それどころか暴走してプログラムをメチャメチャにしてしまうことだってあるのです。

マシン語のプログラムはまず**自分が何をしたいか、今何をしようとしているのか**をはっきりさせて作るように心がけましょう。

プログラム

```
10 WIDTH 40,25:CONSOLE 0,25,0,1
20 DEFINT A-Z :COLOR 2 :CLS
```

10　**画面を40×25モードにして、スクロールを0行から25行まで、ファンクションキーを消して、カラーモードに切り換えます。**

20　**変数を整数型になおしておいてスピードアップをはかり、「赤」を指定しておいて、画面をクリア。**

プログラムの最初によくやる、ごく一般的な初期設定です。

```
30 X=37 :Y=6  :E=1
```

30　Xに37、
Yに6、
Eに1を代入します。

ゲームの主人公であるクローバの位置を、変数XとYによって表すことにします。

主人公の一番最初の居場所を横37、縦6の位置とするわけです。

次にクローバが行かなければならない場所（最初右端にいるクローバは左端に行かねばなりません）を変数Eで表します。

左端へ行かなければならないのでX座標で言うところの1を代入します。

```
40 DEF FNP(X,Y)=PEEK(&HF3C8+X+X+Y*120)
```

40　FNP（X，Y）という関数を宣言します。

X、YにそれぞれX座標Y座標を入れてやると、現在その位置に表示されてあるキャラクタのアスキーコードがわかります。

ただし、40字モード専用です。

```
50 I=INP(0)
```

50　キーボードの状態をIに代入します。

テンキーの、［0］［1］［2］［3］［4］［5］［6］［7］が押されているかいないかが、変数Iに入ります。

テンキー	0	1	2	3	4	5	6	7
INP(0)=	254	253	251	247	239	223	191	127

```
60 V=(I=247)*(X<36)-(I=253)*(X>1)
```

60　Ｖに、Ｉが247で、かつＸが36より小さければ＋１
　　　　Ｉが253で、かつＸが１よりも大きければ－１
　　を代入します。

Ｉが247、つまり 3 を押していて、かつＸ座標が36より小さければ右へ移動することを変数Ｖに代入します。

Ｉが253、つまり 1 を押していて、かつＸ座標が１より大きければ左へ移動することを変数Ｖに代入します。

```
70 X=X+V :P=FNP(X,Y)
```

70　ＸにＸ＋Ｖ、
　　ＰにＦＮＰ（Ｘ，Ｙ）を代入します。

キーボードからの指令によってＶは次に進むべき方向を持っています。そこで、ＸにＶを足してクローバの表示位置を１つ移動させるのです。

そのあと、その場所に何が表示されてあるか変数Ｐに読み込みます。

```
80 IF P=42 THEN 210
```

80　もし、Ｐが42なら210行へ飛ぶ。

読み込んだＰの値が42なら、つまり次に移動しようとしている所がアスタリスクなら210行に飛びます。

この場合、210行へ飛ぶとゲームオーバーになります。

１回でもぶつかると「ゲームオーバー」というわけです。

ちなみに、42は「＊」のアスキーコードです。

```
90 IF X<>E THEN 110
```

90　もし、XとEが違うなら110行へ飛ぶ。

クローバが目的地、つまり右端から出発してまだ左端に到着していなければ、なにもしないで110行へ飛びます。

```
100 Y=Y+2 :E=-35*(E=1)+1 :BEEP 1
```

100　YにY＋2
Eに「−35＊（E＝1）＋1」を代入して、
音を鳴らします。

〈注〉もしEの値が1の場合は、論理式（E＝1）の値は「−1」(真ということ)になりますから、
E＝−35＊（E＝1)＋1
＝−35＊（−1）＋1
＝36
となります。

90行で110行へ飛ばなかった場合、つまり目的地に到着した場合この処理をします。

目的地に着いたら下へ降りなければなりません。Yに2を足してクローバの位置を下へ移動させます。

そして次の目的地を決めます。今のEの値が1なら36、そうでなければ1が代入されるのです。

```
110 COLOR 5 :LOCATE X-V,Y-1:PRINT " ";
120          LOCATE X   ,Y  :PRINT "♣";
```

110　水色を指定し、
X−V，Y−1の場所に
空白を描きます。

120　X，Yの場所に
クローバを描きます。

移動する前の場所のクローバを消して、移動した場所に新しくクローバを描き込み、画面上を移動させます。

```
130 FOR I=1 TO 2
140  LOCATE RND*34+2,24
150  COLOR 2:PRINT "*";
160 NEXT
```

130　Ｉを１から２までくり返します。

140　カーソルを横２～35の乱数、縦24の位置に指定します。

150　「赤」を指定し、
　アスタリスクを表示します。

160　130行からくり返します。

画面の一番下に２個アスクリスクを表示するのです。

```
170 PRINT
```

170　改行します。

画面の一番下にカーソルが来ているので、ここで改行すると画面が１つ上にスクロールします。

```
180 IF Y<22 THEN BEEP 0 :GOTO 50
```

180　もし、Ｙが22より小さければ、音を消して、
50行へ飛びます。

一番下へ降りていなかったら、再びキー入力の段階へもどって、そこからくり返す、というわけです。

```
190 COLOR 6
200 PRINT "Congratulation!":BEEP 0
```

190　「黄」を指定。

200　「Congration！」と書いて、
音を消します。

もし、一番下に来たらこれでゲーム終了にするのです。

```
210 LOCATE 13,10:PRINT (Y-6)*100;"Pts."
```

210　横13、縦10の位置に、
（Ｙ－６）＊100Pts. 表示。

クローバがどれだけ降りたかによってスコアが決まります。

```
220 IF INP(4)<>247 THEN 220
230 RUN
```

220　もし、ＩＮＰ(４)が247でなければ、くり返す。

230　プログラム実行！

Ｓキーを押すまで待ち続け、Ｓを押した途端ゲームを始めます。
と、このようなゲームだったわけです。

6・2　改造

「画面をもっときれいにしたい」
「リアリティのあるゲームにしたい」
「もっと変化がほしい」

いろいろ希望は尽きません。
それでは簡単におもしろくなる改造箇所を探してみましょう。

背景を描く

背景があるのとないのとでは、随分雰囲気も違ってきます。

それに、グラフィックで背景を描けば、ゲームで使用しているのはキャラクタの画面でしたから、ゲーム自体には全然支障をきたしません。当たり前のようなことですが、これがとても効果があるのです。

ただし、これはマシン語を使ってまでするよりＢＡＳＩＣによってする方が適切かと思われます。

アスタリスクの動きを変える

このゲームの命は、なんと言ってもこのアスタリスクです。たくさんのアスタリスクが動いている中を進んでゆくからおもしろいわけで、これが止まっていたりアスタリスクの数が少なかったりしたらおもしろくないでしょう。

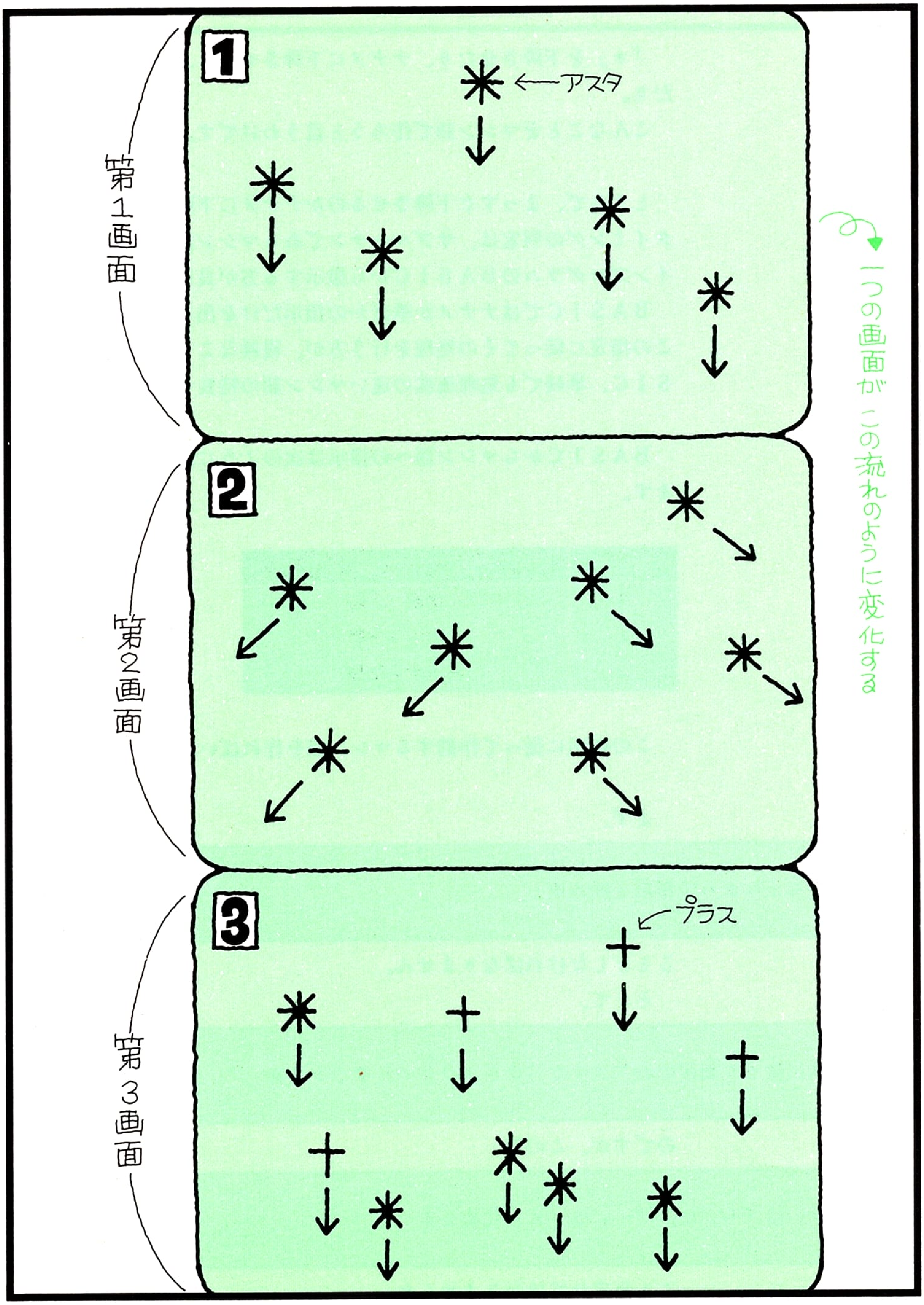
そこでこのアスタリスクの動きを変えてゲームをもっとおもしろくしてやろうというわけです。

1　下から浮き上がってくるというのはどうも不自然ですから、上から降ってくるようにしましょう。

2　ただアスタリスクがまっすぐに落ちてくるだけでは変化がありません。時々、ななめにも落ちるようにしましょう。

3　降って来るアスタリスクが「＊」では平凡ですから、「＋」と交互に表示してキラキラしているようにしてみましょう。

以上の３つのことをマシン語で行うことにします。



6·3 ASTERISK

「＊」を下降させたり、ナナメに下降させたり、キラキラ点滅させたり。

こんなことをマシン語で作ろうと言うわけです。

ところで、まっすぐ下降させるのかナナメに下降させるのかというタイミングの判定は、サブルーチンであるマシン語でするよりも、メインプログラムのＢＡＳＩＣから指示する方が良いと思います。

ＢＡＳＩＣではナナメか垂直かの指示だけを出し、マシン語の方でこの指定に従ってその処理を行う方が、複雑なことを得意とするＢＡＳＩＣ、単純でも処理速度の速いマシン語の特長を生かせるからです。

ＢＡＳＩＣからマシン語への指示は次のように決めておくことにします。

指　示	内　　容
0	垂直　下降
1	左ナナメ下降
2	右ナナメ下降

この指示に従って作動するマシン語を作ればいいわけです。

まず、

ＢＡＳＩＣからの指示を読み取る

ことをしなければなりません。

そして、

指示に従って画面中のアスタリスクをスクロールダウンする

のですが、この時、

「＊」は「＋」に、「＋」は「＊」に変える

ことを忘れてはなりませんね。

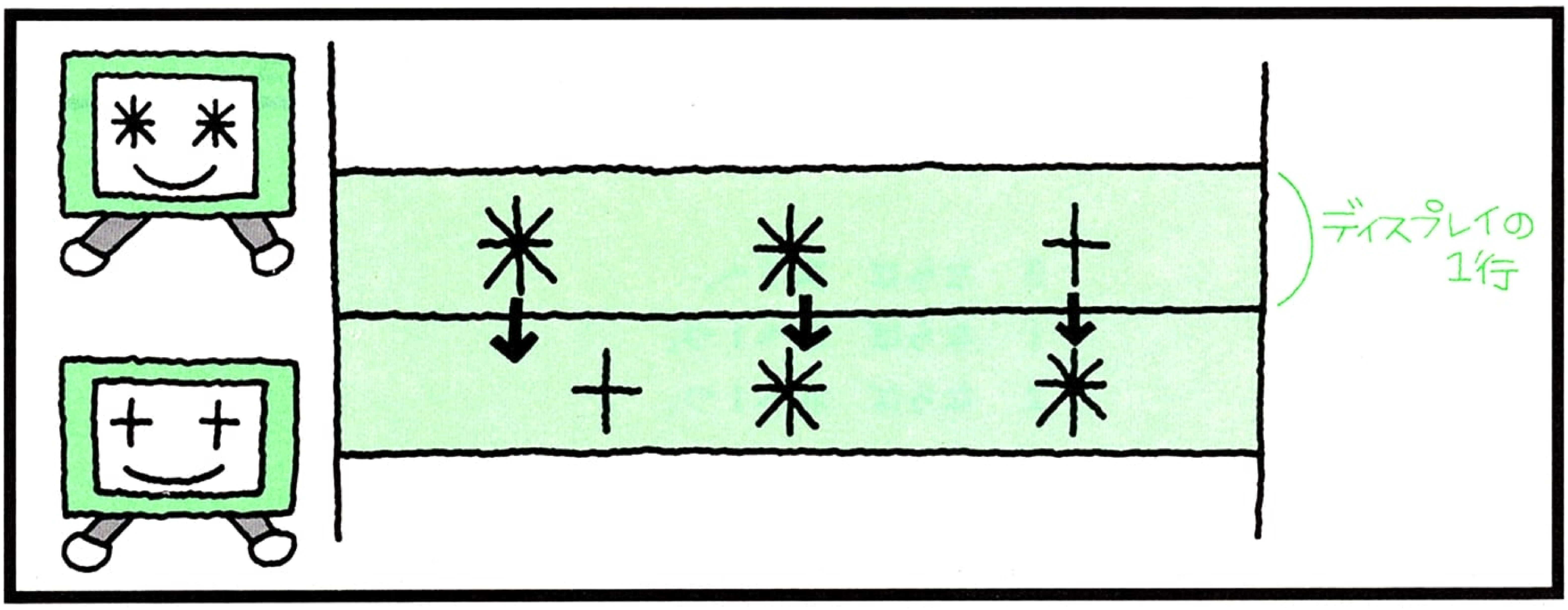
スクロールダウン(混沌)

ＢＡＳＩＣからのデータを読み取るのはわかるとして、「＊，＋」のスクロールダウンが問題です。

いくらマシン語だからといっても一度に画面全部のキャラクタを調べるのは無理な話で、やはり面倒でも**１つ１つ調べてゆかなくてはならない**のです。

調べると言っても好き勝手に調べていったのでは見落としが出て来るかも知れませんから端から順番に調べましょう。
そして、それも**下から順番に調べる**のです。

なぜなら、もし上からスクロールダウンしてゆくと、その下の段のキャラクタは上からスクロールダウンして来たキャラクタによって消されてしまうからです。



スクロールダウンする時は下から、スクロールアップする時は上から処理してゆくようにします。
また、スクロールダウンする時、１つ横にずらしてゆくと全体としてナナメにスクロールすることになります。

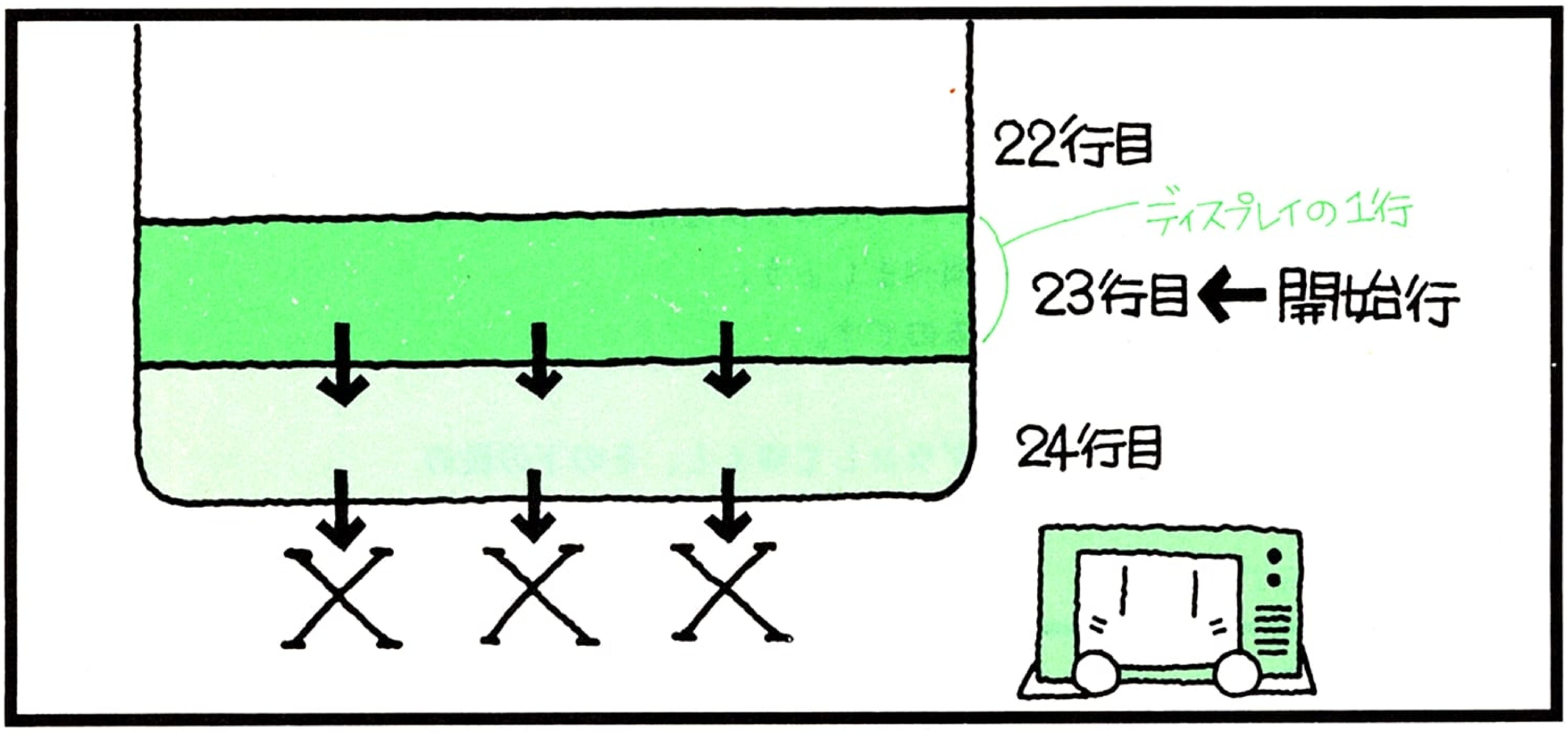
今回のマシン語は、**アスタリスクのスクロールが重要**だというわけです。

スクロールダウン（考察）

まずＢＡＳＩＣからデータを受け取ります。

次にスクロールダウンします。

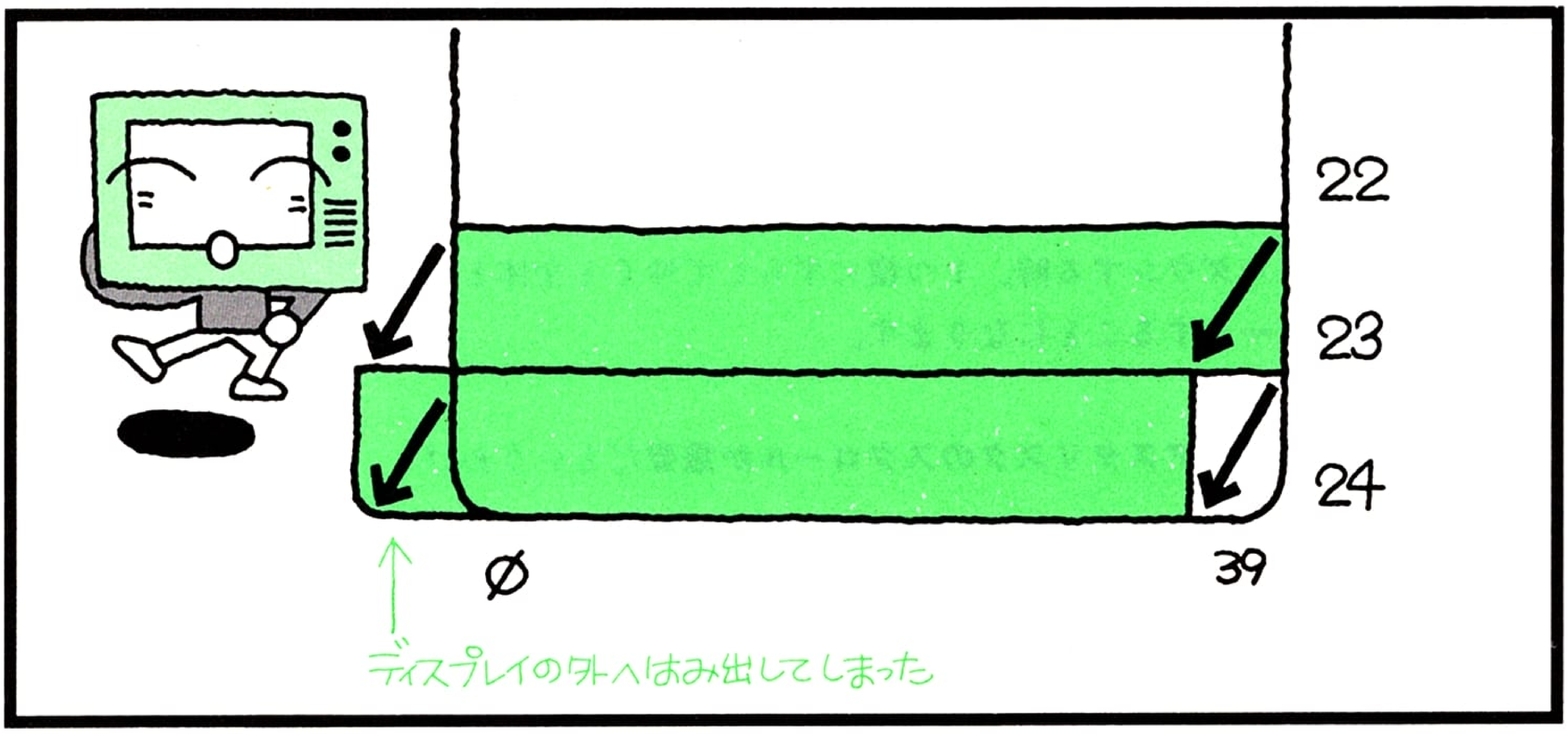
一番下の段はスクロールダウンする必要がありませんから、下から2段目よりスクロールダウン開始です。



この時、ＢＡＳＩＣからの情報か、

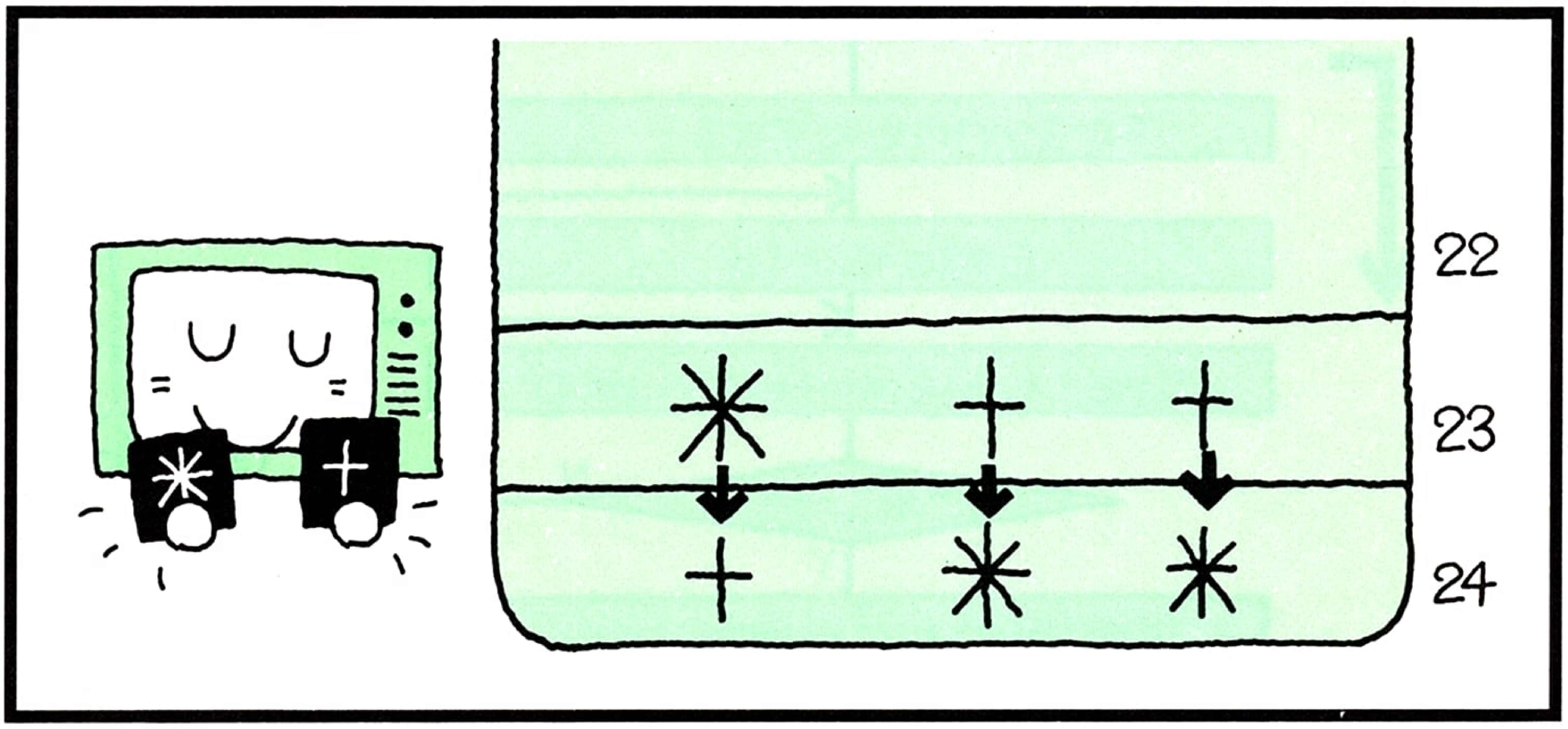
0　ならば　真下へ、
1　ならば　左へ1つ、
2　ならば　右へ1つ、

ずらします。

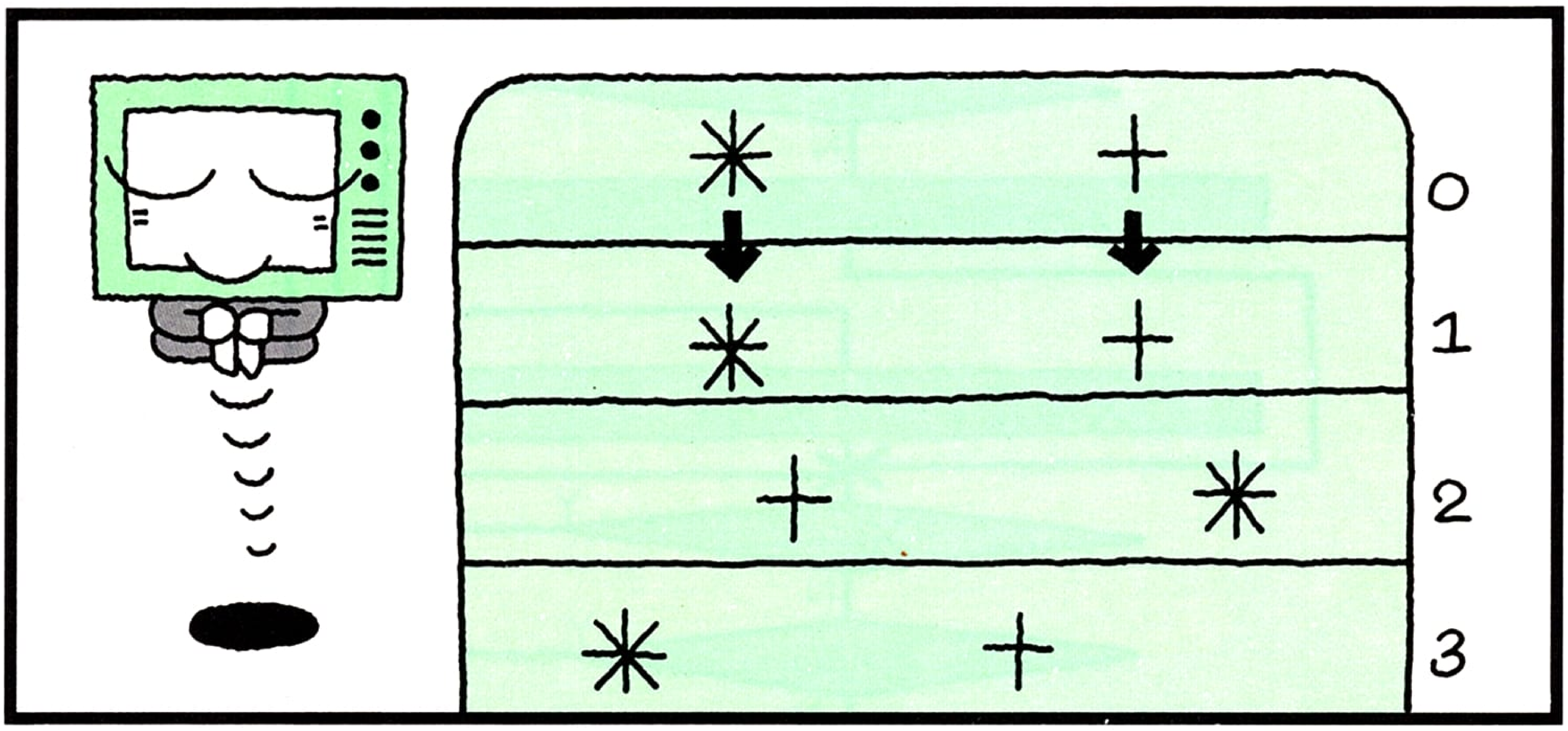


ナナメにずらす時、画面からはみ出た所には書き込めません。
書き込めないわけでもありませんが、下手をして**暴走させてしまってはなんにもならない**からです。

X座標が－1や40になった所は書き込む処理を飛ばすのです。
スクロールダウンは「＊」と「＋」についてのみ行います。
そして、**スクロールした時に「＊」ならば「＋」に、「＋」ならば「＊」に変換**します。



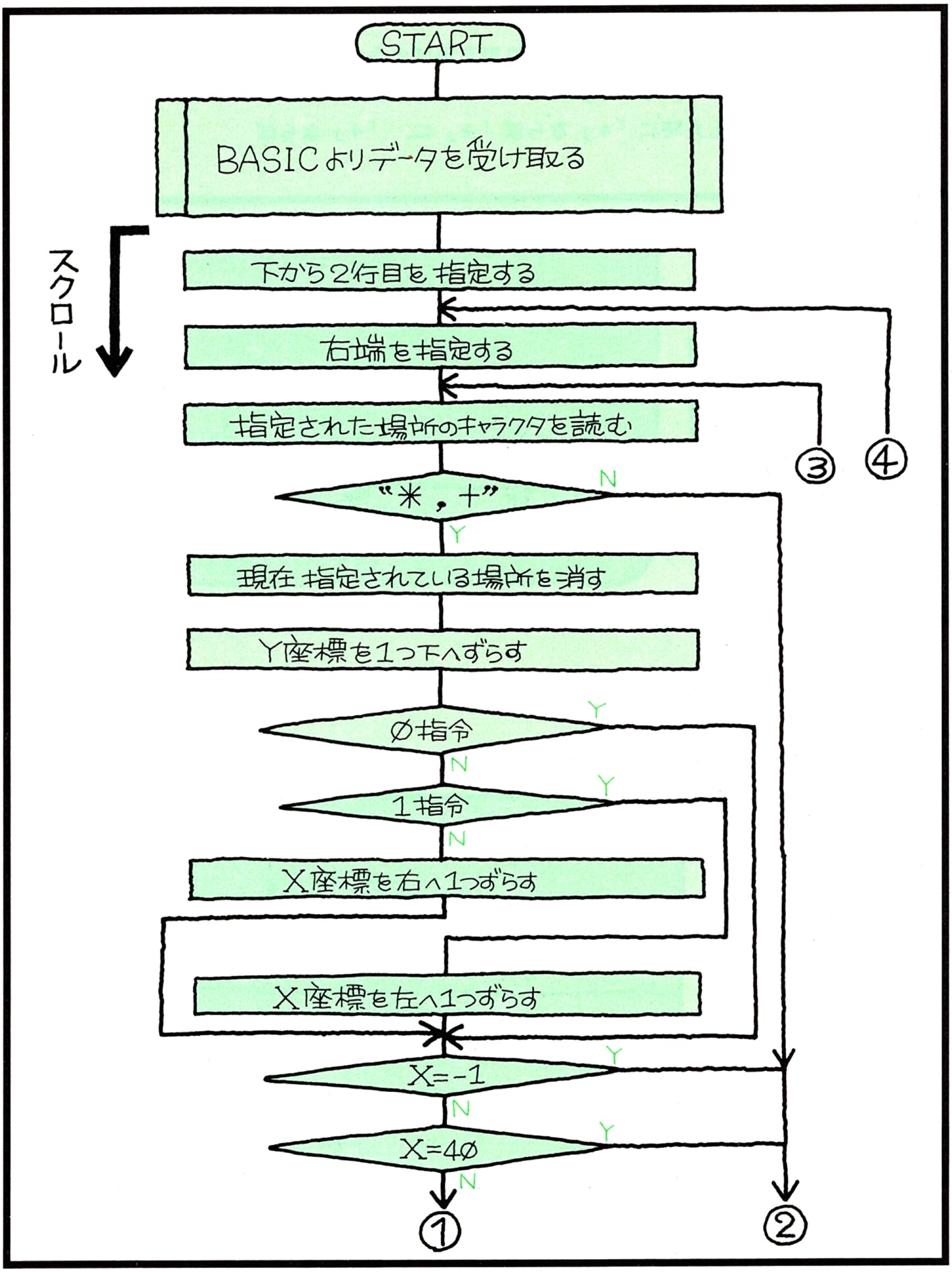
一段ずつスクロールする行を上げていって一番上の行までくり返します。

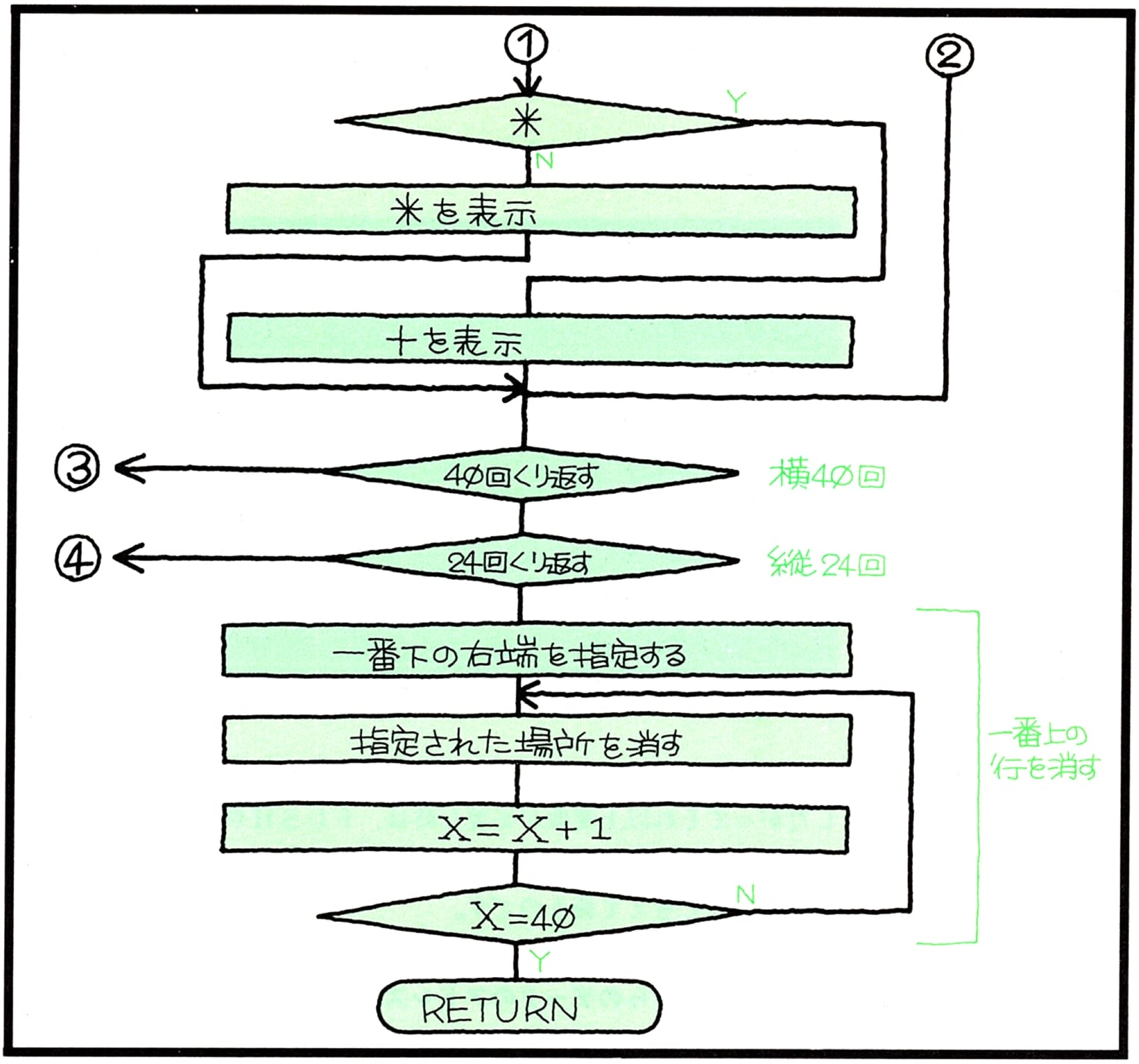


最後に１番下の行にあるキャラクタを全部消しておしまいです。
あとはＢＡＳＩＣに戻ってゲームを続けるだけです。

スクロールダウン（計画）

フローチャート化してみます。





必要になる変数を最初から書き出してみます。

ＢＡＳＩＣからのデータアドレス用	2バイト
ＢＡＳＩＣからのデータ用	1バイト
座標指定用のＸ	1バイト
座標指定用のＹ	1バイト
ＶＲＡＭ座標指定用	2バイト
キャラクタ格納用	1バイト
キャラクタ消去用	1バイト
書き込み用のＸ	1バイト
書き込み用のＹ	1バイト
キャラクタ表示用	1バイト
条件判断用	1バイト

スクロールダウン（変数）

重用な変数の順に並べてみます。

重用な変数　条件判断用	1バイト
ＢＡＳＩＣからのデータ用	1バイト
座標指定用のＸ	1バイト
座標指定用のＹ	1バイト
キャラクタ格納用	1バイト
書き込み用のＸ	1バイト
書き込み用のＹ	1バイト
ＶＲＡＭ座標指定用	2バイト
キャラクタ消去用	1バイト
キャラクタ表示用	1バイト
ＢＡＳＩＣからのデータアドレス用	2バイト

マシン語はＢＡＳＩＣと違って使える変数が８つしかありません。したがってそれ以上変数が必要な時は、ＰＵＳＨやＰＯＰを使うか、もしくは使用したあともう使う必要がなくなったレジスタを新たに別のレジスタと考えて使うのです。

ＢＡＳＩＣからのデータのアドレスを入れるレジスタはＨＬレジスタです。これは、一度メモリからデータを読み取ってしまえばもう不用ですから次の処理で必要な時、自由に使用してかまいません。

条件判断用のレジスタはＡレジスタが便利です。いつでも使うことができるようにあけておきましょう。

ＢＡＳＩＣからの指令を入れておくレジスタにはＢレジスタを使います。

座標指定用のＸ，Ｙの数値を代入しておくのは、Ｄ，Ｅレジスタにします。

画面から読み取ったキャラクタを保管しておくのはＣレジスタにします。

画面表示用のX、Y座標を入れておくのは、H、Lレジスタにします。

VRAM座標がわかれば画面表示の座標を代入しているレジスタはもう不用ですし、HLにXY座標のデータを入れて「4290」番地をCALLすればHLにVRAM座標が入るわけですから、これも、HLレジスタを使用することにします。

消去や表示キャラクタを代入するレジスタ。これはほんの一瞬使用するだけですからAレジスタを使います。

これで、レジスタを何に使用するか決まりました。

A	:	条件判断用 キャラクタ消去用 キャラクタ表示用
B	:	BASICからの指令データ
C	:	画面から読み取ったキャラクタ
D	:	座標指定用のXデータ
E	:	座標指定用のYデータ
H	:	表示位置指定用のXデータ
L	:	表示位置指定用のYデータ
HL	:	BASICからのデータアドレス用 VRAM座標指定用

スクロールダウン（製作）

それでは、プログラムの製作をします。

スタート番地は、B900にしてみましょう。

最初はBASICからの指令を読み出します。この時、指定は0か1か2ですので、下2ケタの1バイトだけを読み出してやればいいのです。メモリには下2ケタ上2ケタの順で格納されていますので最初の1バイトをBレジスタに取り出します。

アドレス	マシン語	ラベル	ニーモニック	コメント
			ORG B900	START
				B←指令
B900	46	START	LD B, (HL)	

下から2行目で右端の座標を指定し、そのVRAMの座標を計算します。

B901	1E 17		LD E,23(17H)	E←Yデータ
B903	16 00	JUMP4	LD D,0	D←Xデータ
B905	62	JUMP3	LD H,D	HL←VRAM座標（B905～B909）
B906	6B		LD L,E	
B907	24		INC H	
B908	2C		INC L	
B909	CD 9D 42		CALL 429DH	

指定された場所のキャラクタを読み取り、「＊」か「＋」ならば次の処理へ行き、そうでなければ飛ばします。

B90C	4E		LD C, (HL)	C←キャラクタ
B90D	79		LD A, C	
B90E	FE 2A		CP 2AH	C＝"＊"?
B910	CA 1B B9		JP Z, NEXT1	
B913	FE 2B		CP 2BH	C＝"＋"?
B915	CA 1B B9		JP Z, NEXT1	
B918	C3 4F B9		JP JUMP2	
B91B		NEXT1		

指定されている場所をＳＰＡＣＥで消して、表示位置のＹ座標を１つ下へずらします。

B91B	3E 20	NEXT1	LD A, 20H	A←"␣"
B91D	77		LD (HL), A	(HL)←A
B91E	62		LD H, D	H←X
B91F	6B		LD L, E	L←Y＋1
B920	2C		INC L	

ＢＡＳＩＣからの指令が０ならば何もせず、１ならば左へ１つ、そうでなければ右へ１つずらします。

B921	78		LD A, B	A←指令
B922	FE 00		CP 0	A=0 ?
B924	CA 31 B9		JP Z, NEXT2	
B927	FE 01		CP 1	A=1 ?
B929	CA 30 B9		JP Z, LEFT	
B92C	24		INC H	H=X+1
B92D	C3 31 B9		JP NEXT2	
B930	25	LEFT	DEC H	H=X−1
B931		NEXT2		

Ｘが－１や40になった時は、表示処理を飛ばします。

B931	7C	NEXT2	LD A, H	A←X
B932	FE FF		CP −1(0FFH)	A=−1 ?
B934	CA 4F B9		JP Z, JUMP2	
B937	FE 28		CP 40 (28H)	A=40 ?
B939	CA 4F B9		JP Z, JUMP2	

ＶＲＡＭ座標計算を行い、表示されてあったキャラクタが“＊”なら“＋”を、そうでなければ“＊”を表示します。

B93C	24		INC H	VRAM座標計算
B93D	2C		INC L	
B93E	CD 9D 42		CALL 429DH	
B941	79		LD A, C	A←キャラクタ
B942	FE 2A		CP A, 2AH	A＝“＊”?
B944	CA 4C B9		JP Z, PLUS	
B947	3E 2A		LD A, 2AH	A←“＊”
B949	C3 4E B9		JP PRINT	
B94C	3E 2B	PLUS	LD A, 2BH	A←“＋”
B94E	77	PRINT	LD (HL), A	表示!!

あとは、これを横に40回、縦に24回くり返します。

B94F	14	JUMP2	INC D	D←X＋1
B950	7A		LD A, D	D<>41 ?
B951	FE 29		CP 41 (29H)	
B953	C2 05 B9		JP NZ, JUMP3	
B956	1D		DEC E	E<Y−1
B957	7B		LD A, E	E<>−1 ?
B958	FE FF		CP −1(0FFH)	
B95A	C2 03 B9		JP NZ, JUMP4	

最後に１番下の行を消してＢＡＳＩＣに戻ります。

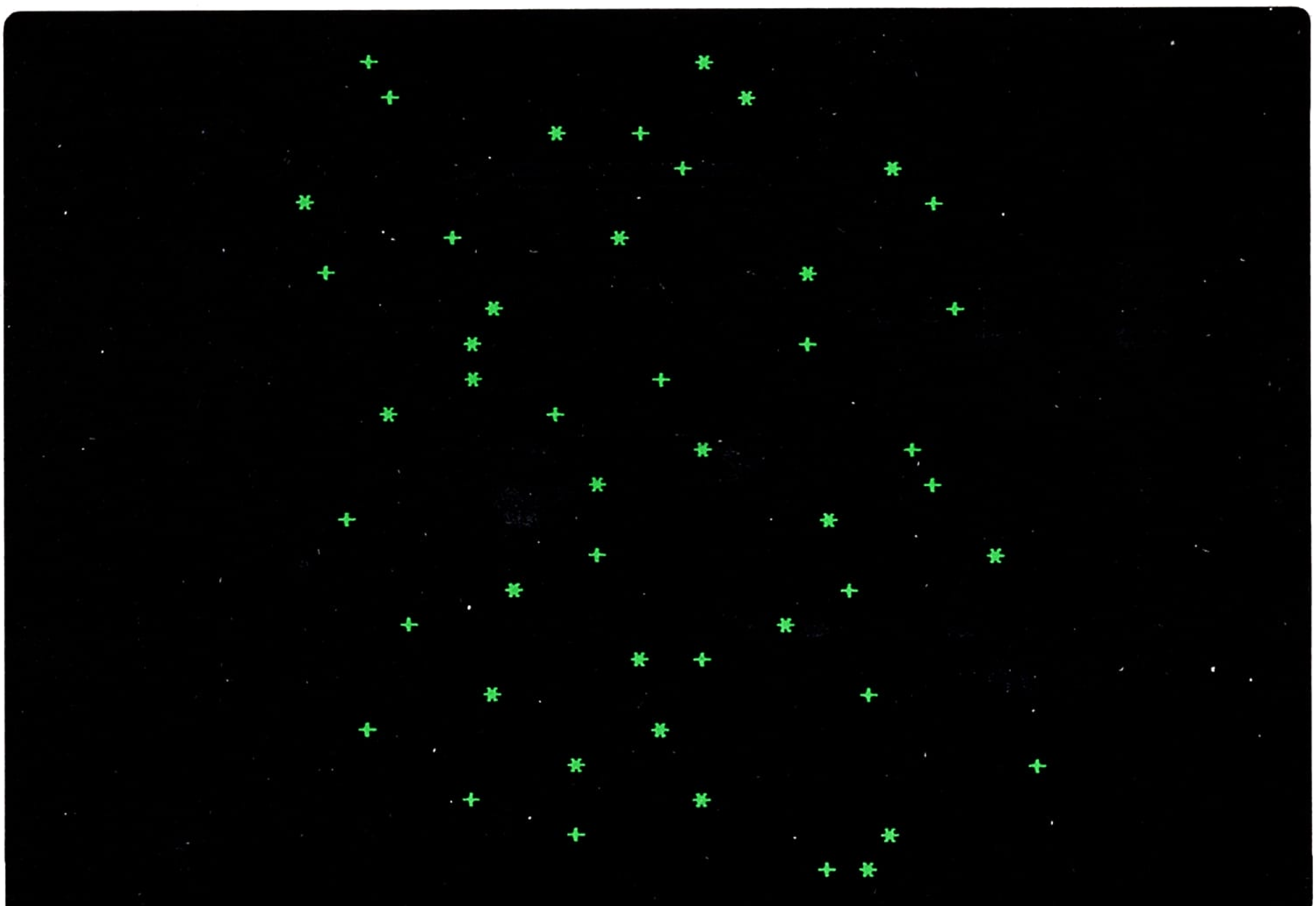
B95D	16 000		LD D,0	D←Xデータ
B95F	1E 18		LD E,24(18)	E←Yデータ
B961	62	JUMP5	LD H,D	URAM座標計算
B962	6B		LD L,E	
B963	24		INC H	
B964	2C		INC L	
B965	CD 9D 42		CALL 429DH	
B968	3E 20		LD A,20H	A←"␣"
B96A	77		LD (HL), A	(HL)←A
B96B	14		INC D	D←X+1
B96C	7A		LD A, D	D<>40 ?
B96D	FE 28		CP 40 (28H)	
B96F	C2 61 B9		JP NZ, JUMP5	
B972	C9		RET	RETURN

以上でマシン語部は完成です。

動くかどうかためしてみましょう。

```
10 WIDTH 40,25 :CONSOLE 0,25,0,1
20 DEFINT A-Z
30 DEF USR=&HB900
40 LOCATE RND*39, 0:PRINT "*";
50 LOCATE RND*39, 0:PRINT "+";
60 Z=INT(RND*3)
70 A=USR(Z)
80 GOTO 40
```

RUNします。



多少ゆっくりな気もしますが、ちゃんと動いてくれました。

ゆっくりなのは、**ROM内のVRAM計算ルーチンを使用したため**です。しかし、これでもけっこうおもしろいゲームになりそうです。

6·4 プログラムの完成

それでは、完成したマシン語プログラムを使ってゲームを改造してみます。

ＡＳＴＥＲＩＳＫは上から降って来ますので、もとのゲームとは逆に下から上へ登るゲームにします。

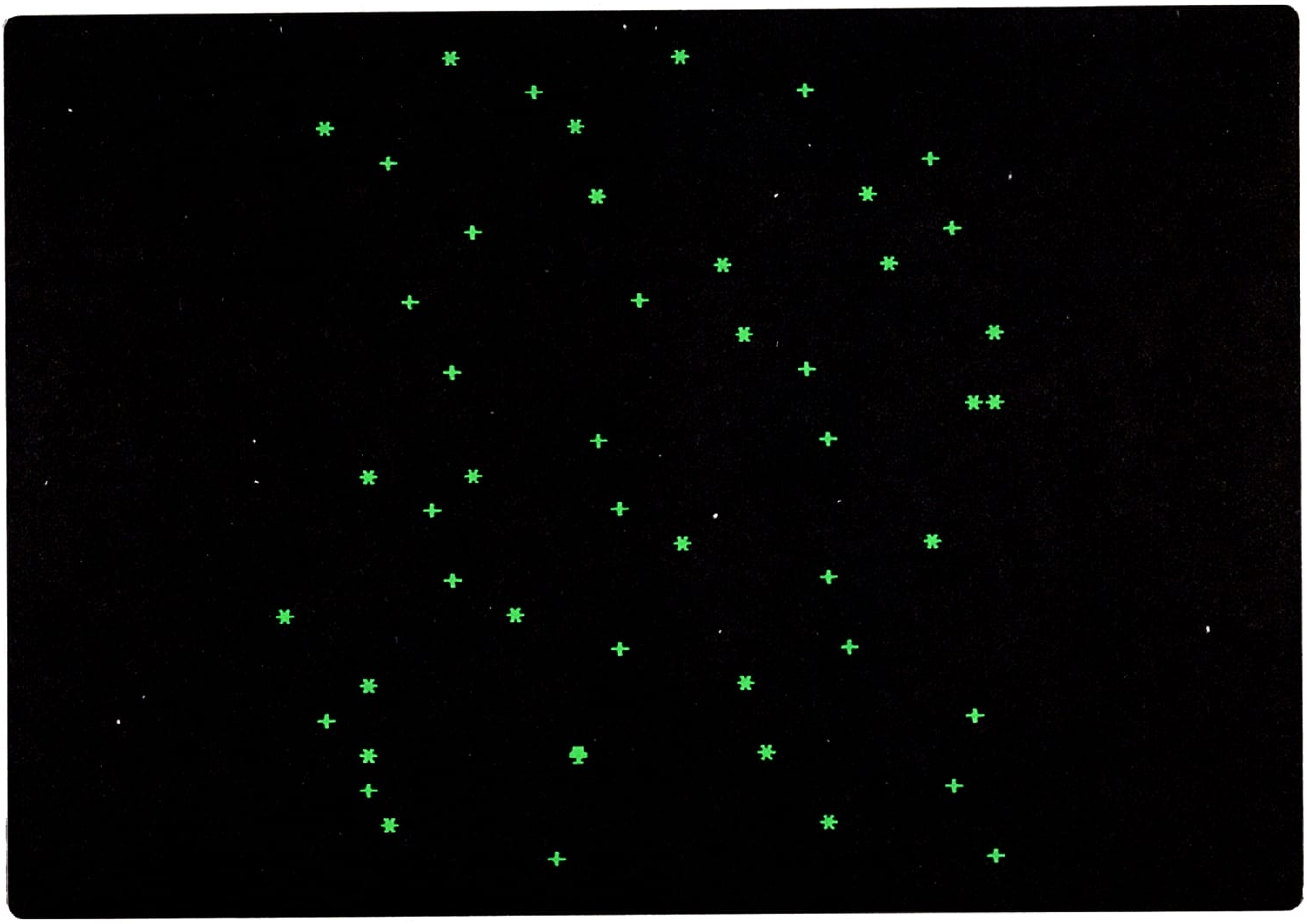
BASIC

```
10 WIDTH 40,25:CONSOLE 0,25,0,1
20 DEFINT A-Z :COLOR 2 :CLS
25 DEF USR=&HB900
30 X=37 :Y=22 :E=1 :W=0
40 DEF FNP(X,Y)=PEEK(&HF3C8+X+X+Y*120)
50 I=INP(0)
60 V=(I=247)*(X<36)-(I=253)*(X>1)
70 X=X+V :P=FNP(X,Y)
80 IF P=42 THEN 210
90 IF X<>E THEN W=0 :GOTO 110
100 Y=Y-2 :W=2 :E=-35*(E=1)+1 :BEEP 1
110 COLOR 2 :LOCATE X-V,Y+W:PRINT " ";
120 COLOR 5 :LOCATE X  ,Y  :PRINT "♣";
130 FOR I=1 TO 2
140  LOCATE RND*34+2,0
150  COLOR 2:PRINT "*";
160 NEXT
170 Z=INT(RND*3):A=USR(Z)
180 IF Y>6 THEN BEEP 0 :GOTO 50
190 COLOR 6
200 PRINT "Congratulation!":BEEP 0
210 LOCATE 13,10:PRINT (22-Y)*100;"Pts.
220 IF INP(4)<>247 THEN 220
230 RUN
```

マシン語

```
B900 46 1E 17 16 00 62 6B 24   F     bk$
B908 2C CD 9D 42 4E 79 FE 2A   ,ヘ、BNy▒*
B910 CA 1B B9 FE 2B CA 1B B9   ハ ケ▒+ハ ケ
B918 C3 4F B9 3E 20 77 62 6B   テOケ> wbk
B920 2C 78 FE 00 CA 31 B9 FE   ,x▒ ハ1ケ▒
B928 01 CA 30 B9 24 C3 31 B9    ハ0ケ$テ1ケ
B930 25 7C FE FF CA 4F B9 FE   %¦▒▒ハOケ▒
B938 28 CA 4F B9 24 2C CD 9D   (ハOケ$,ヘ、
B940 42 79 FE 2A CA 4C B9 3E   By▒*ハLケ>
B948 2A C3 4E B9 3E 2B 77 14   *テNケ>+w
B950 7A FE 29 C2 05 B9 1D 7B   z▒)ツ ケ {
B958 FE FF C2 03 B9 16 00 1E   ▒▒ツ ケ
B960 18 62 6B 24 2C CD 9D 42    bk$,ヘ、B
B968 3E 20 77 14 7A FE 28 C2   > w z▒(ツ
B970 61 B9 C9                  aケノ
```

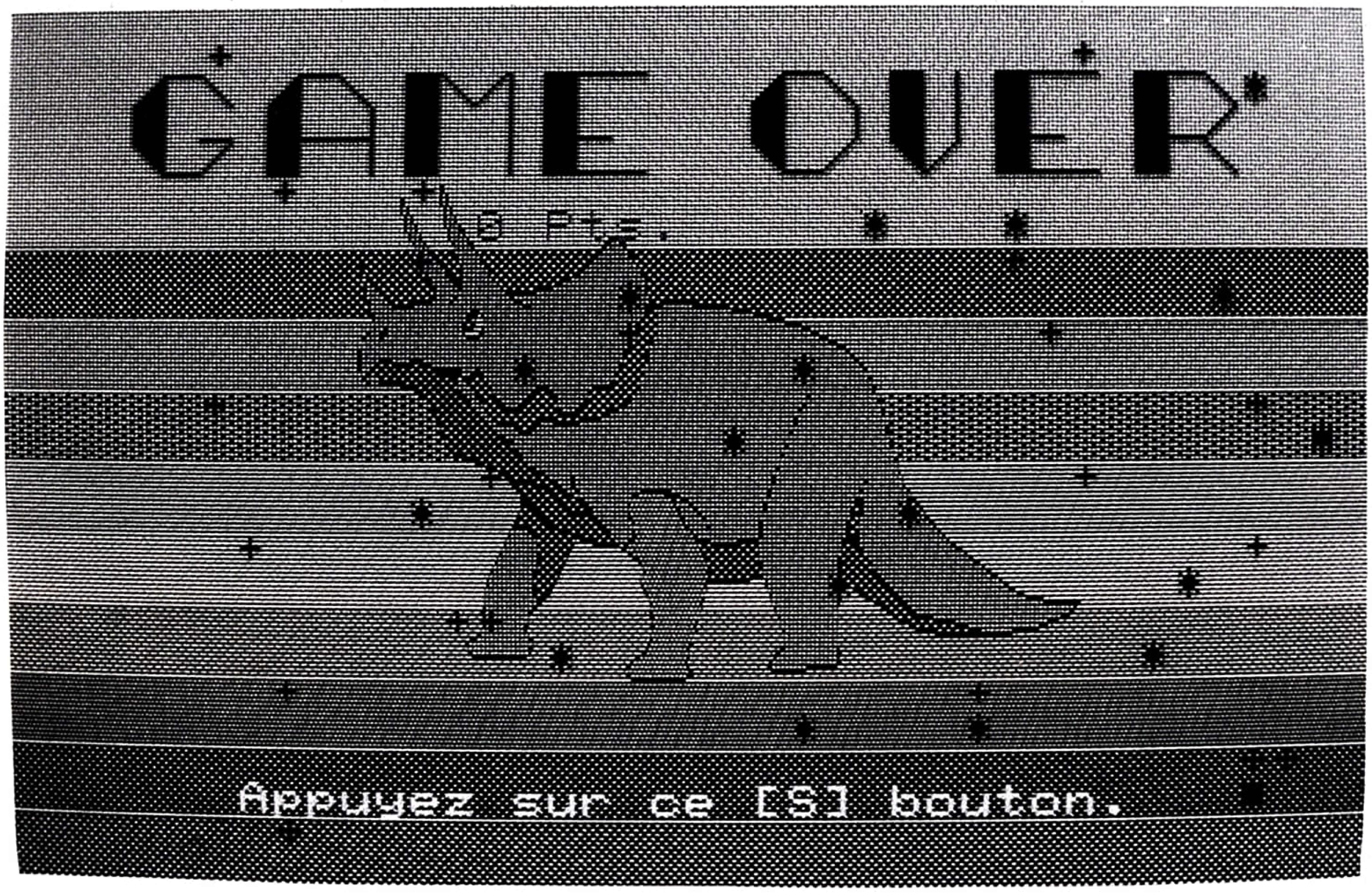
実行画面

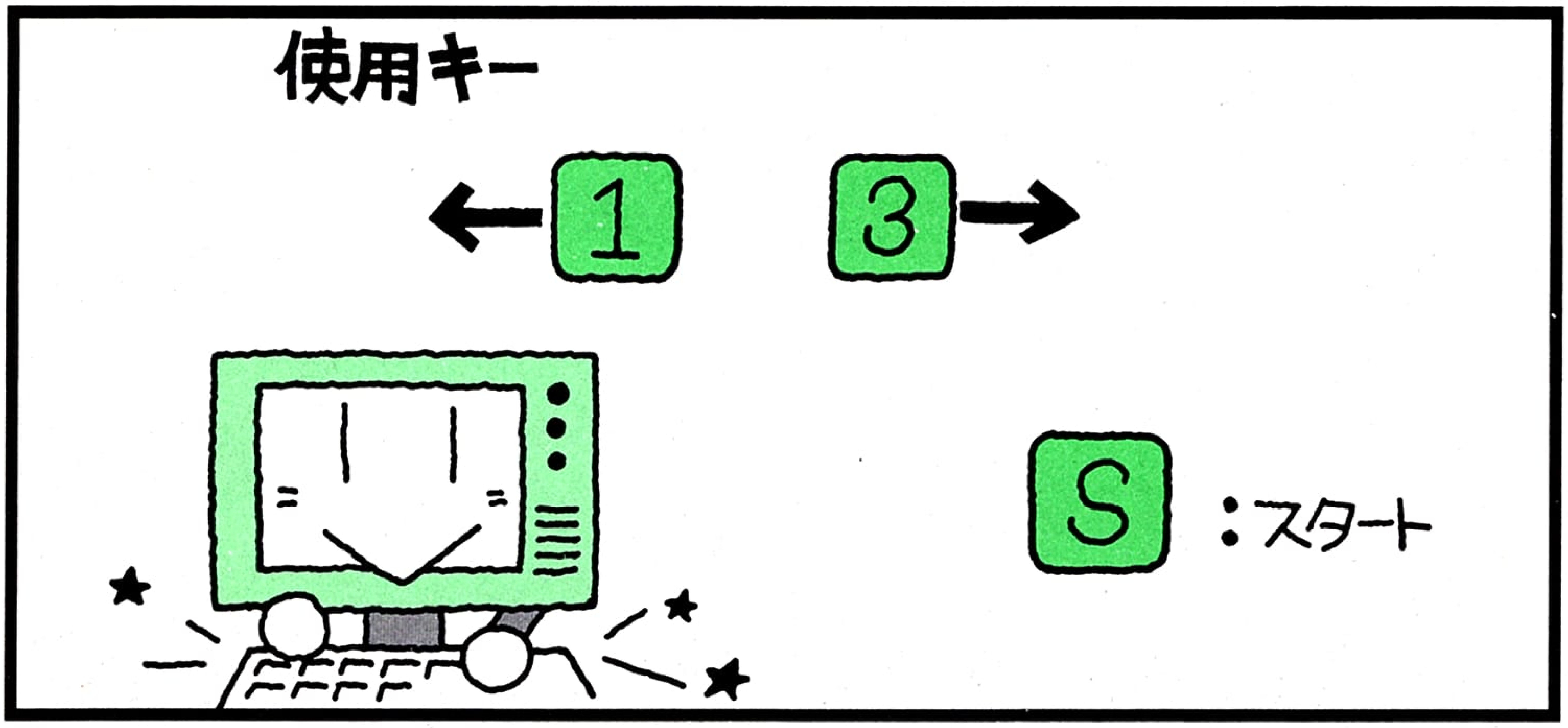


6・5 プログラム完了

最後に、バックにこってみます。グラフィックスを描いたうえで、このゲームを実行させるのです。

「トリケラトプス」の登場です。





PROGRAM LIST

```
10 ' ┌──────────────────────────────────┐
20 '│                                  │
30 '│   <<< ASTERISK ??? >>>           │
40 '│                                  │
50 '│  MACHINE : B900 - B97F           └
60 '│   BASIC  :   10 - 2640           ┌
70 '│      Y.MORI    1984.6.23         │
80 ' └──────────────────────────────────┘
90 '
100 CLEAR ,&HB8FF
110 DEF USR=&HB900 :' SCROLL DOWN
120 '
130 GOSUB *INIT.
140 '
150 GOSUB *STAGE
160 '
170 REM ■■■■■   MAIN LOOP   ■■■■■
180 '
190 GOSUB *MAIN
200 '
210 GOSUB *ASTERISK
220 '
230 IF FLAG$="OVER" THEN *OVER
240 IF FLAG$="MISS" THEN *MISS
250 '
260 GOTO 190
270 '
280 REM ■■■■■   M A I N !   ■■■■■
290 '
300 *MAIN
310  I=INP(0)
320  V=(I=247)*(X<36)*2-(I=253)*(X>1)*2
330  X=X+V
340  P=FNP(X,Y)
350  '
360  IF P=42 OR P=43 THEN FLAG$="MISS"
370  IF X<>E THEN W=0 :GOTO 400
380  '
```

Notes beside the listing:

- Lines 100–110: マシン語設定
- Line 130: イニシャライズ
- Line 150: 画面表示
- Lines 190–260: メインループ
- Lines 310–340: キースキャン

```
390  BEEP 1 :Y=Y-2 :W=2 :E=-36*(E=1)+1
400  COLOR 2:LOCATE X-V,Y+W :PRINT " ";   ┐
410  COLOR 0:LOCATE X  ,Y   :PRINT "♣";   ┘─MOVE
420  '
430  IF Y<7 THEN FLAG$="OVER"  ←── 最上階
440  '
450  BEEP 0
460  '
470  RETURN
480  '
490 REM ■■■■■  ASTERISK!  ■■■■■
500 '
510 *ASTERISK
520  FOR I=1 TO 2              ┐
530   LOCATE RND*34+2, 0       ├─アスタリスク
540   COLOR 2 :PRINT "*";      │
550  NEXT                      ┘
560  Z=INT(RND*3)  ←── 方向
570  '
580  A=USR(Z)  ←── スクロールダウン
590  '
600  RETURN
610  '
620 REM ■■■■■  GAME OVER  ■■■■■
630 '
640 *OVER
650  COLOR 6 :LOCATE 10, 3               ┐
660  PRINT "Congraturation!!"            ├─最上階
670  '                                   ┘
680 REM ■■■■■  ENDING!  ■■■■■
690 '
700 *MISS
710  COLOR 6
720  FOR CH=135 TO 128 STEP -1
730   FOR I=1 TO 10                        ┐
740    LOCATE  X, Y :PRINT CHR$(CH);       │
750    BEEP 1                              ├─FLASH!
760    LOCATE  X, Y :PRINT " ";            │
770    BEEP 0                              │
780   NEXT                                 ┘
```

```
790  NEXT
800  COLOR 1 :LOCATE 13, 6
810  PRINT (22-Y)*100;"Pts."
820  COLOR 2                                          ┐
830  LOCATE  4, 2 :PRINT "◢◣ ◢◣ ■◣◢ ■▀              │
 ◢◣ ■| ■▀ ■◣"                                        │
840  LOCATE  4, 3 :PRINT "■ _ ■▄▌■ |■▄                │
     ■ |■| ■▄ ■▄◢"                                   ├ ゲーム
850  LOCATE  4, 4 :PRINT "◥◢ ■ |■ |■▄                 │  オーバー
_ ◥◢ ◥◢ ■▄ ■ ◥"                                      │
860  COLOR 4                                         │
870  LOCATE  7,22 :PRINT "Appuyez sur ce              │
 [S] bouton."                                        ┘
880  '
890  IF INP(4)<>247 THEN 890 ← キー入力待ち
900  '
910  GOSUB 1640 ← 初期設定
920  GOSUB 1750 ← ベル
930  '
940  GOTO 170
950  '
1000 REM ■■■■■ INITIALIZE! ■■■■■
1010 '
1020 *INIT.
1030  WIDTH 40,25 :CONSOLE 0,25,0,1                  ┐
1040  DEFINT A-Z  :COLOR 2 :CLS 3                     │
1050  DEF FNP(X,Y)=PEEK(&HF3C8+X+X+Y*120              ├ 初期設定
)                                                    │
1060  DIM T$(17)                                      ┘
1070  '
1080  FOR I=1 TO 17
1090   FOR J=1 TO 2                                   ┐
1100    READ B$,R$,G$                                 │
1110    B=VAL("&H"+B$)                                │
1120    R=VAL("&H"+R$)                                ├ 中間色作り
1130    G=VAL("&H"+G$)                                │
1140    T$(I)=T$(I)+CHR$(B)+CHR$(R)+CHR$              │
(G)                                                  │
1150   NEXT                                           │
1160  NEXT                                            ┘
```

```
1170  '
1180  ST=1 ←―――――――――――――――― タイトルフラグ
1190  '
1200  RETURN
1210  '
1220 REM ▬▬▬▬▬  S T A G E  ▬▬▬▬▬
1230 '
1240 *STAGE
1250  FOR I=7 TO 23 STEP 2
1260   LINE(0,I*8)-(640,I*8),7          ――― 横線
1270  NEXT
1280 '
1290  FOR I=2 TO 10
1300   PAINT(0,I*16+25),T$(I),7         ――― カラーPAINT
1310  NEXT
1320 '
1330  RESTORE 2220
1340  FOR I=1 TO 310
1350   READ X,Y,Z
1360   IF Z=1 THEN LINE(X,Y)-(X,Y),1    ――― 「トリケラトフス」青
1370   IF Z=0 THEN LINE      -(X,Y),1
1380  NEXT
1390  '
1400  RESTORE 2530
1410  FOR I=1 TO 13
1420   READ X,Y,Z                       ――― 「トリケラトフス」PAINT
1430   PAINT(X,Y),T$(Z),1
1440  NEXT
1450  '
1460 '
1470  RESTORE 2220
1480  FOR I=1 TO 310
1490   READ X,Y,Z
1500   IF Z=1 THEN LINE(X,Y)-(X,Y),7    ――― 「トリケラトフス」白
1510   IF Z=0 THEN LINE      -(X,Y),7
1520  NEXT
1530  '
1540 PAINT(0,0),T$(1),7 ←――――――――――― 空
1550  '
1560  '
```

```
1570  RESTORE 2220
1580  FOR I=1 TO 310
1590   READ X,Y,Z
1600   IF Z=1 THEN LINE(X,Y)-(X,Y),0
1610   IF Z=0 THEN LINE     -(X,Y),0
1620  NEXT
1630  '
1640  X=37 :Y=22 :E=1 :W=0 :FLAG$=""
1650  '
1660  IF ST=0 THEN RETURN
1670  '
1680  COLOR 6
1690  LOCATE  4, 2:PRINT "◢▀◣ ◢█◣ ▀█▀ █▀
▔ █▀◣ █ ◢█◣ █ ╱  █
1700  LOCATE  4, 3:PRINT "█▄▄▌ ◥█◤  █  █▄
 █▄╱ █ ◥█◣ █◤   █
1710  LOCATE  4, 4:PRINT "█  ▏◥█◤  █  █▄
▁ █ ╲ █ ◥█◤ █▀◣  ▁
1720  LOCATE  7,22 :PRINT "Appuyez sur c
e [S] bouton."
1730  IF INP(4)<>247 THEN 1730
1740  '
1750  FOR I=1 TO 100
1760   BEEP I MOD 4
1770  NEXT
1780  COLOR 2 :CLS
1790  ST=0
1800  '
1810  RETURN
1820  '
2000 REM ▬▬▬▬  D A T A !  ▬▬▬▬
2010 '
2020 DATA ff,00,55 ,ff,00,55
2030 DATA 00,00,44 ,00,00,11
2040 DATA 00,00,aa ,00,00,aa
2050 DATA 00,08,a2 ,00,80,2a
2060 DATA 00,55,ff ,00,55,ff
2070 DATA 44,11,ff ,11,44,ff
2080 DATA 00,55,bb ,00,55,ee
2090 DATA 00,ff,aa ,00,ff,55
```

(1570–1620) 「トリケラトプス」黒

(1640) 初期設定

(1680–1730) タイトル

(1750–1780) BEEP

(2020–2090) カラーデータ

```
2100 DATA 00,aa,22 ,00,aa,88
2110 DATA 88,00,22 ,22,00,88
2120 DATA 00,08,a2 ,00,80,2a
2130 DATA 00,41,aa ,00,14,55
2140 DATA 00,88,88 ,00,22,22
2150 DATA 00,80,80 ,00,08,08
2160 DATA 00,00,88 ,00,00,22
2170 DATA 00,22,aa ,00,88,aa
2180 DATA ff,ff,ff ,ff,ff,ff
2190 '
2200 REM █████  L I N E !  █████
2210 '
2220 DATA 304, 69, 1, 316, 68, 0, 336, 6
9, 0, 349, 70, 0, 378, 74, 0, 393, 78 ,
0, 401, 81, 0, 405, 84, 0, 411, 91, 0, 4
16, 100, 0
2230 DATA 422, 109, 0, 428, 113, 0, 439,
 120, 0, 455, 126, 0, 468, 131, 0, 478,
134, 0, 494, 135, 0, 502, 135, 0, 495, 1
38, 0, 480, 141, 0
2240 DATA 463, 142, 0, 443, 142, 0, 429,
 141, 0, 418, 138, 0, 409, 135, 0, 399,
132, 0, 392, 130, 0, 405, 107, 1, 399, 1
13, 0, 395, 119, 0
2250 DATA 392, 126, 0, 392, 131, 0, 391,
 141, 0, 392, 148, 0, 390, 149, 0, 363,
150, 0, 359, 149, 0, 363, 146, 0, 367, 1
44, 0, 367, 142, 0
2260 DATA 362, 136, 0, 357, 129, 0, 355,
 124, 0, 356, 117, 0, 361, 109, 0, 367,
101, 0, 374, 92, 0, 378, 89, 0, 357, 129
, 1, 345, 130, 0
2270 DATA 332, 130, 0, 314, 110, 1, 319,
 115, 0, 331, 125, 0, 332, 129, 0, 330,
134, 0, 325, 142, 0, 322, 147, 0, 322, 1
49, 0, 325, 151, 0
2280 DATA 325, 151, 1, 318, 152, 0, 296,
 152, 0, 293, 150, 0, 296, 148, 0, 304,
145, 0, 306, 141, 0, 305, 134, 0, 301, 1
31, 0, 302, 128, 0
```

「トリケラトプス」

```
2290 DATA 301, 126, 0, 290, 122, 0, 276,
 113, 0, 270, 109, 0, 270, 107, 0, 290,
122, 1, 283, 122, 0, 275, 121, 0, 266, 1
19, 0, 260, 118, 0
2300 DATA 246, 111, 0, 238, 106, 0, 227,
 104, 0, 217, 102, 0, 211, 99, 0, 207 ,
94, 0, 204, 90, 0, 199, 88, 0, 188, 86,
0, 183, 86, 0
2310 DATA 181, 85, 0, 179, 83, 0, 179, 8
2, 0, 272, 120, 1, 268, 124, 0, 260, 132
, 0, 249, 138, 0, 248, 141, 0, 251, 144,
 0, 250, 146, 0
2320 DATA 246, 147, 0, 224, 147, 0, 221,
 146, 0, 220, 144, 0, 222, 143, 0, 226,
142, 0, 228, 140, 0, 229, 132, 0, 231, 1
29, 0, 243, 117, 0
2330 DATA 246, 114, 0, 246, 111, 0, 242,
 118, 1, 248, 118, 0, 252, 120, 0, 255,
123, 0, 253, 127, 0, 251, 131, 0, 253, 1
34, 0, 256, 134, 0
2340 DATA 219, 84, 1, 212, 84, 0, 206, 8
3, 0, 203, 82, 0, 199, 81, 0, 196, 81 ,
0, 193, 82, 0, 189, 82, 0, 186, 81, 0, 1
81, 81, 0
2350 DATA 178, 83, 0, 175, 85, 0, 176, 8
6, 0, 172, 83, 0, 172, 80, 0, 174, 76 ,
0, 177, 74, 0, 178, 72, 0, 177, 70, 0, 1
74, 65, 0
2360 DATA 172, 61, 0, 172, 58, 0, 174, 6
1, 0, 178, 64, 0, 183, 67, 0, 188, 69 ,
0, 192, 69, 0, 200, 68, 0, 204, 68, 0, 2
07, 66, 0
2370 DATA 200, 60, 0, 194, 53, 0, 190, 4
6, 0, 190, 44, 0, 195, 49, 0, 200, 54 ,
0, 211, 61, 0, 216, 64, 0, 218, 64, 0, 2
13, 68, 1
2380 DATA 217, 66, 0, 219, 63, 0, 217, 5
9, 0, 212, 53, 0, 210, 48, 0, 208, 41 ,
0, 216, 50, 0, 224, 56, 0, 235, 64, 0, 2
42, 67, 0
```

```
2390 DATA 250, 69, 0, 240, 66, 1, 246, 6
6, 0, 259, 65, 0, 271, 62, 0, 278, 59 ,
0, 283, 56, 0, 287, 54, 0, 291, 54, 0, 3
01, 61, 0
2400 DATA 301, 61, 1, 304, 65, 0, 305, 6
8, 0, 303, 71, 0, 300, 73, 0, 301, 76 ,
0, 300, 77, 0, 294, 79, 0, 292, 82, 0, 2
89, 85, 0
2410 DATA 286, 87, 0, 280, 87, 0, 278, 8
8, 0, 273, 92, 0, 267, 93, 0, 264, 92 ,
0, 262, 89, 0, 261, 88, 0, 256, 87, 0, 2
53, 87, 0
2420 DATA 250, 89, 0, 247, 91, 0, 243, 9
2, 0, 240, 92, 0, 235, 90, 0, 228, 84 ,
0, 225, 83, 0, 219, 85, 1, 227, 84, 0, 2
19, 83, 1
2430 DATA 209, 83, 0, 221, 85, 1, 225, 8
8, 0, 229, 91, 0, 232, 93, 0, 237, 95 ,
0, 242, 95, 0, 244, 96, 0, 247, 97, 0, 2
51, 97, 0
2440 DATA 254, 95, 0, 257, 94, 0, 259, 9
4, 0, 264, 95, 0, 280, 94, 0, 294, 91 ,
0, 300, 87, 0, 303, 83, 0, 306, 77, 0, 3
09, 74, 0
2450 DATA 309, 69, 0, 241, 108, 1, 250,
108, 0, 256, 109, 0, 264, 113, 0, 268 ,
114, 0, 271, 115, 0, 274, 115, 0, 279, 1
15, 0, 279, 115, 0
2460 DATA 214, 83, 1, 218, 88, 0, 228, 9
4, 0, 235, 98, 0, 242, 99, 0, 248, 104,
0, 265, 109, 0, 269, 111, 0, 275, 113, 0
, 292, 123, 1
2470 DATA 296, 121, 0, 298, 120, 0, 298,
 119, 0, 295, 115, 0, 295, 114, 0, 298,
111, 0, 302, 110, 0, 311, 110, 0, 316, 1
12, 0, 315, 111, 1
2480 DATA 318, 111, 0, 325, 113, 0, 329,
 115, 0, 333, 119, 0, 333, 119, 0, 325,
120, 1, 329, 121, 0, 336, 122, 0, 355, 1
21, 0, 403, 109, 1
```

```
2490 DATA 402, 114, 0, 401, 118, 0, 400,
 122, 0, 395, 120, 1, 398, 121, 0, 402,
123, 0, 407, 127, 0, 419, 133, 0, 429, 1
38, 0, 441, 139, 0
2500 DATA 454, 141, 0, 459, 142, 0, 185,
 74, 1, 183, 74, 0, 181, 75, 0, 181, 76,
 0, 183, 76, 0, 185, 74, 0, 185, 75, 0,
221, 73, 1
2510 DATA 221, 72, 0, 223, 70, 0, 226, 7
0, 0, 229, 71, 0, 229, 73, 0, 228, 75 ,
0, 224, 76, 0, 222, 76, 0, 220, 75, 0, 2
21, 73, 0
2520 DATA 221, 73, 1, 221, 74, 0, 223, 7
4, 0, 225, 73, 0, 225, 72, 0, 224, 71 ,
0, 222, 71, 0, 220, 73, 0, 220, 73, 0, 2
20, 75, 0
2530 REM ■■■■■  P A I N T  ■■■■■
2540 '
2550 DATA 250, 80,16 ,300,100,16
2560 DATA 220,100, 9 ,240,140,16
2570 DATA 250,115,13 ,260,115,13
2580 DATA 320,140,12 ,430,140,13
2590 DATA 250, 95,15 ,222, 73,13
2600 DATA 350,125,13 ,398,120,15
2610 DATA 226, 73,17 ,330,120,15
2620 '┌─────────────────────────────┐
2630 '│ FORESIGHT PLANNING SECTION  │
2640 '└─────────────────────────────┘
```

PAINTデータ

ディスクへのセーブ方法

186～187ページのプログラムを例に説明します。

まず186ページのベーシックプログラムを入力し、次のようにディスクにセーブします。

```
SAVE "ASTERI" ⏎
```

次に187ページのマシン語をメモリにセットします。マシン語モニタのＳコマンドを使ってＢ９００番地から次のように機械語をセットします。

```
MON ⏎
h]SB900 ⏎
B900 FF-46■
```

上の画面は、Ｂ９００番地の内容ＦＦを４６にセットしようとしているところです。ここでスペースキーを押してください。すると次にＢ９０１番地の内容「００－」が表示されますから、同じように１Ｅをキーインしスペースキーを押してください。以後、Ｂ９７２番地まで同様の操作をくり返して187ページのマシン語をセットします。

次にマシン語のセーブです。次のように入力します。

```
BSAVE "asteri",&HB900,115 ⏎
```

115：プログラムのバイト数

&HB900：先頭アドレス

これでマシン語プログラムが「asteri」というファイル名でディスクにセーブできました。プログラムのバイト数は「＆ＨＢ９７２－＆ＨＢ９００＋１」で計算できます。

この状態でＲＵＮさせれば187ページの実行画面を得ることができます。なお、後日あらためてプログラムをディスクからロードして実行したいときは次のようにキーインしてください。

```
BLOAD "asteri" ⏎
RUN "ASTERI" ⏎
```

第7章

Z80インストラクション活用表

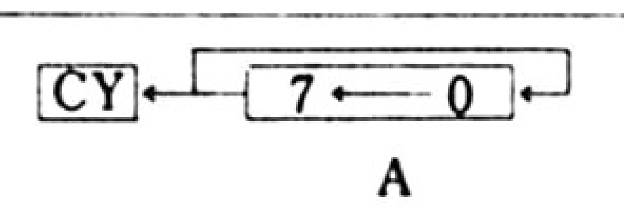
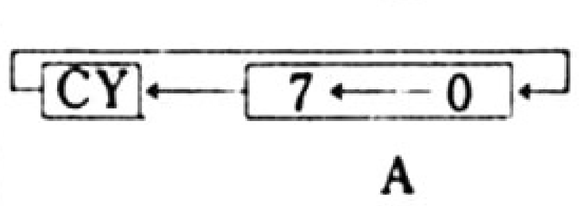
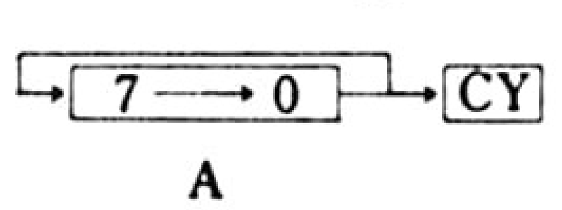
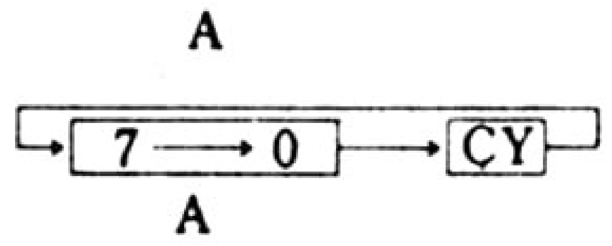
Z80 インストラクション・セット

命令群	ニーモニック	オペレーション	フラグ S Z H P/V N C	バイト	ステート	OPコード 76 543 210	
8ビット・ロード命令	LD r, r′	r←r′	• • • • • • •	1	4	01 r r′	Ⓔ
	LD r, n	r←n	• • • • • • •	2	7	00 r 110 ← n →	Ⓔ
	LD r, (HL)	r←(HL)	• • • • • • •	1	7	01 r 110	Ⓔ
	LD r, (IX+d)	r←(IX+d)	• • • • • • •	3	19	11 011 101 01 r 110 ← d →	Ⓔ
	LD r, (IY+d)	r←(IY+d)	• • • • • • •	3	19	11 111 101 01 r 110 ← d →	Ⓔ
	LD (HL), r	(HL)←r	• • • • • • •	1	7	01 110 r	Ⓔ
	LD (IX+d), r	(IX+d)←r	• • • • • • •	3	19	11 011 101 01 110 r ← d →	Ⓔ
	LD (IY+d), r	(IY+d)←r	• • • • • • •	3	19	11 111 101 01 110 r ← d →	Ⓔ
	LD (HL), n	(HL)←n	• • • • • • •	2	10	00 110 110 ← n →	
	LD (IX+d), n	(IX+d)←n	• • • • • • •	4	19	11 011 101 00 110 110 ← d → ← n →	
	LD (IY+d), n	(IY+d)←n	• • • • • • •	4	19	11 111 101 00 110 110 ← d → ← n →	
	LD A, (BC)	A←(BC)	• • • • • • •	1	7	00 001 010	
	LD A, (DE)	A←(DE)	• • • • • • •	1	7	00 011 010	
	LD A, (nn)	A←(nn)	• • • • • • •	3	13	00 111 010 ← n → ← n →	
	LD (BC), A	(BC)←A	• • • • • • •	1	7	00 000 010	
	LD (DE), A	(DE)←A	• • • • • • •	1	7	00 010 010	
	LD (nn), A	(nn)←A	• • • • • • •	3	13	00 110 010 ← n → ← n →	
	LD A, I	A←I	↕ ↕ 0 IFF 0 •	2	9	11 101 101 01 010 111	
	LD A, R	A←R	↕ ↕ 0 IFF 0 •	2	9	11 101 101 01 011 111	
	LD I, A	I←A	• • • • • • •	2	9	11 101 101 01 000 111	
	LD R, A	R←A	• • • • • • •	2	9	11 101 101 01 001 111	
16ビット・ロード命令	LD dd, nn	dd←nn	• • • • • • •	3	10	00 dd0 001 ← n → ← n →	Ⓐ
	LD IX, nn	IX←nn	• • • • • • •	4	14	11 011 101 00 100 001 ← n → ← n →	
	LD IY, nn	IY←nn	• • • • • • •	4	14	11 111 101 00 100 001 ← n → ← n →	
	LD HL, (nn)	H←(nn+1) L←(nn)	• • • • • • •	3	16	00 101 010 ← n → ← n →	
	LD dd, (nn)	dd_H←(nn+1) dd_L←(nn)	• • • • • • •	4	20	11 101 101 01 dd1 011 ← n → ← n →	Ⓐ

命令群	ニーモニック	オペレーション	S	Z	H	P/V	N	C	バイト	ステート	76	543	210	
16ビット・ロード命令	LD IX, (nn)	$IX_H \leftarrow (nn+1)$ $IX_L \leftarrow (nn)$	・	・	・	・	・	・	4	20	11 00 ← ←	011 101 n n	101 010 → →	
	LD IY, (nn)	$IY_H \leftarrow (nn+1)$ $IY_L \leftarrow (nn)$	・	・	・	・	・	・	4	20	11 00 ← ←	111 101 n n	101 010 → →	
	LD (nn), HL	$(nn+1) \leftarrow H$ $(nn) \leftarrow L$	・	・	・	・	・	・	3	16	00 ← ←	100 n n	010 → →	
	LD (nn), dd	$(nn+1) \leftarrow dd_H$ $(nn) \leftarrow dd_L$	・	・	・	・	・	・	4	20	11 01 ← ←	101 dd0 n n	101 011 → →	Ⓐ
	LD (nn), IX	$(nn+1) \leftarrow IX_H$ $(nn) \leftarrow IX_L$	・	・	・	・	・	・	4	20	11 00 ← ←	011 100 n n	101 010 → →	
	LD (nn), IY	$(nn+1) \leftarrow IY_H$ $(nn) \leftarrow IY_L$	・	・	・	・	・	・	4	20	11 00 ← ←	111 100 n n	101 010 → →	
	LD SP, HL	SP←HL	・	・	・	・	・	・	1	6	11	111	001	
	LD SP, IX	SP←IX	・	・	・	・	・	・	2	10	11 11	011 111	101 001	
	LD SP, IY	SP←IY	・	・	・	・	・	・	2	10	11 11	111 111	101 001	
	PUSH qq	$(SP-2) \leftarrow qq_L$ $(SP-1) \leftarrow qq_H$	・	・	・	・	・	・	1	11	11	qq0	101	Ⓑ
	PUSH IX	$(SP-2) \leftarrow IX_L$ $(SP-1) \leftarrow IX_H$	・	・	・	・	・	・	2	15	11 11	011 100	101 101	
	PUSH IY	$(SP-2) \leftarrow IY_L$ $(SP-1) \leftarrow IY_H$	・	・	・	・	・	・	2	15	11 11	111 100	101 101	
	POP qq	$qq_H \leftarrow (SP+1)$ $qq_L \leftarrow (SP)$	・	・	・	・	・	・	1	10	11	qq0	001	Ⓑ
	POP IX	$IX_H \leftarrow (SP+1)$ $IX_L \leftarrow (SP)$	・	・	・	・	・	・	2	14	11 11	011 100	101 001	
	POP IY	$IY_H \leftarrow (SP+1)$ $IY_L \leftarrow (SP)$	・	・	・	・	・	・	2	14	11 11	111 100	101 001	
エクスチェンジ命令	EX DE, HL	DE↔HL	・	・	・	・	・	・	1	4	11	101	011	
	EX AF, AF'	AF↔AF'	・	・	・	・	・	・	1	4	00	001	000	
	EXX	(BC↔BC' DE↔DE' HL↔HL')	・	・	・	・	・	・	1	4	11	011	001	
	EX (SP), HL	H↔(SP+1) L↔(SP)	・	・	・	・	・	・	1	19	11	100	011	
	EX (SP), IX	$IX_H \leftrightarrow (SP+1)$ $IX_L \leftrightarrow (SP)$	・	・	・	・	・	・	2	23	11 11	011 100	101 011	
	EX (SP), IY	$IY_H \leftrightarrow (SP+1)$ $IY_L \leftrightarrow (SP)$	・	・	・	・	・	・	2	23	11 11	111 100	101 011	
ブロック転送命令	LDI	(DE)←(HL) DE←DE+1 HL←HL+1 BC←BC−1	・	・	0	① ↕	0	・	2	16	11 10	101 100	101 000	
	LDIR	(DE)←(HL) DE←DE+1 HL←HL+1 BC←BC−1 until BC=0	・	・	0	0	0	・	2	21 if BC≠0 16 if BC=0	11 10	101 110	101 000	

命令群	ニーモニック	オペレーション	S	Z	H	P/V	N	C	バイト	ステート	OPコード 76 543 210
ブロック転送命令	LDD	(DE)←(HL) DE←DE−1 HL←HL−1 BC←BC−1	•	•	0	①↕	0	•	2	16	11 101 101 10 101 000
	LDDR	(DE)←(HL) DE←DE−1 HL←HL−1 BC←BC−1 until BC=0	•	•	0	0	0	•	2	21 if BC≠0 16 if BC=0	11 101 101 10 111 000
ブロック・サーチ命令	CPI	A−(HL) HL←HL+1 BC←BC−1	↕	②↕	↕	①↕	1	•	2	16	11 101 101 10 100 001
	CPIR	A−(HL) HL←HL+1 BC←BC−1 until A=(HL) or BC=0	↕	②↕	↕	①↕	1	•	2	21 if BC≠0 and A≠(HL) 16 if BC=0 or A=(HL)	11 101 101 10 110 001
	CPD	A−(HL) HL←HL−1 BC←BC−1	↕	②↕	↕	①↕	1	•	2	16	11 101 101 10 101 001
	CPDR	A−(HL) HL←HL−1 BC←BC−1 until A=(HL) or BC=0	↕	②↕	↕	①↕	1	•	2	21 if BC≠0 and A≠(HL) 16 if BC=0 or A=(HL)	11 101 101 10 111 001
8ビット算術論理演算命令	ADD A, r	A←A+r	↕	↕	↕	V	0	↕	1	4	10 000 r Ⓔ
	ADD A, n	A←A+n	↕	↕	↕	V	0	↕	2	7	11 000 110 ← n →
	ADD A, (HL)	A←A+(HL)	↕	↕	↕	V	0	↕	1	7	10 000 110
	ADD A, (IX+d)	A←A+(IX+d)	↕	↕	↕	V	0	↕	3	19	11 011 101 10 000 110 ← d →
	ADD, A, (IY+d)	A←A+(IY+d)	↕	↕	↕	V	0	↕	3	19	11 111 101 10 000 110 ← d →
	ADC A, r	A←A+r+CY	↕	↕	↕	V	0	↕	1	4	10 001 r Ⓔ
	ADC A, n	A←A+n+CY	↕	↕	↕	V	0	↕	2	7	11 001 110 ← n →
	ADC A, (HL)	A←A+(HL)+CY	↕	↕	↕	V	0	↕	1	7	10 001 110
	ADC A, (IX+d)	A←A+(IX+d)+CY	↕	↕	↕	V	0	↕	3	19	11 011 101 10 001 110 ← d →
	ADC A, (IY+d)	A←A+(IY+d)+CY	↕	↕	↕	V	0	↕	3	19	11 111 101 10 001 110 ← d →
	SUB r	A←A−r	↕	↕	↕	V	1	↕	1	4	10 010 r Ⓔ
	SUB n	A←A−n	↕	↕	↕	V	1	↕	2	7	11 010 110 ← n →
	SUB (HL)	A←A−(HL)	↕	↕	↕	V	1	↕	1	7	10 010 110
	SUB (IX+d)	A←A−(IX+d)	↕	↕	↕	V	1	↕	3	19	11 011 101 10 010 110 ← d →
	SUB (IY+d)	A←A−(IY+d)	↕	↕	↕	V	1	↕	3	19	11 111 101 10 010 110 ← d →
	SBC A, r	A←A−r−CY	↕	↕	↕	V	1	↕	1	4	10 011 r · Ⓔ
	SBC A, n	A←A−n−CY	↕	↕	↕	V	1	↕	2	7	11 011 110 ← n →
	SBC A, (HL)	A←A−(HL)−CY	↕	↕	↕	V	1	↕	1	7	10 011 110

命令群	ニーモニック	オペレーション	S	Z	H	P/V	N	C	バイト	ステート	OPコード 76 543 210	
8ビット算術論理演算命令	SBC A, (IX+d)	A←A−(IX+d)−CY	↕	↕	↕	V	1	↕	3	19	11 011 101 10 011 110 ← d →	
	SBC A, (IY+d)	A←A−(IY+d)−CY	↕	↕	↕	V	1	↕	3	19	11 111 101 10 011 110 ← d →	
	AND r	A←A∧r	↕	↕	1	P	0	0	1	4	10 100 r	Ⓔ
	AND n	A←A∧n	↕	↕	1	P	0	0	2	7	11 100 110 ← n →	
	AND (HL)	A←A∧(HL)	↕	↕	1	P	0	0	1	7	10 100 110	
	AND (IX+d)	A←A∧(IX+d)	↕	↕	1	P	0	0	3	19	11 011 101 10 100 110 ← d →	
	AND (IY+d)	A←A∧(IY+d)	↕	↕	1	P	0	0	3	19	11 111 101 10 100 110 ← d →	
	OR r	A←A∨r	↕	↕	0	P	0	0	1	4	10 110 r	Ⓔ
	OR n	A←A∨n	↕	↕	0	P	0	0	2	7	11 110 110 ← n →	
	OR (HL)	A←A∨(HL)	↕	↕	0	P	0	0	1	7	10 110 110	
	OR (IX+d)	A←A∨(IX+d)	↕	↕	0	P	0	0	3	19	11 011 101 10 110 110 ← d →	
	OR (IY+d)	A←A∨(IY+d)	↕	↕	0	P	0	0	3	19	11 111 101 10 110 110 ← d →	
	XOR r	A←A∀r	↕	↕	0	P	0	0	1	4	10 101 r	Ⓔ
	XOR n	A←A∀n	↕	↕	0	P	0	0	2	7	11 101 110 ← n →	
	XOR (HL)	A←A∀(HL)	↕	↕	0	P	0	0	1	7	10 101 110	
	XOR (IX+d)	A←A∀(IX+d)	↕	↕	0	P	0	0	3	19	11 011 101 10 101 110 ← d →	
	XOR (IY+d)	A←A∀(IY+d)	↕	↕	0	P	0	0	3	19	11 111 101 10 101 110 ← d →	
	CP r	A−r	↕	↕	↕	V	1	↕	1	4	10 111 r	Ⓔ
	CP n	A−n	↕	↕	↕	V	1	↕	2	7	11 111 110 ← n →	
	CP (HL)	A−(HL)	↕	↕	↕	V	1	↕	1	7	10 111 110	
	CP (IX+d)	A−(IX+d)	↕	↕	↕	V	1	↕	3	19	11 011 101 10 111 110 ← d →	
	CP (IY+d)	A−(IY+d)	↕	↕	↕	V	1	↕	3	19	11 111 101 10 111 110 ← d →	
	INC r	r←r+1	↕	↕	↕	V	0	•	1	4	00 r 100	Ⓔ
	INC (HL)	(HL)←(HL)+1	↕	↕	↕	V	0	•	1	11	00 110 100	
	INC (IX+d)	(IX+d)←(IX+d)+1	↕	↕	↕	V	0	•	3	23	11 011 101 00 110 100 ← d →	
	INC (IY+d)	(IY+d)←(IX+d)+1	↕	↕	↕	V	0	•	3	23	11 111 101 00 110 100 ← d →	
	DEC r	r←r−1	↕	↕	↕	V	1	•	1	4	00 r 101	Ⓔ
	DEC (HL)	(HL)←(HL)−1	↕	↕	↕	V	1	•	1	11	00 110 101	
	DEC (IX+d)	(IX+d)←(IX+d)−1	↕	↕	↕	V	1	•	3	23	11 011 101 00 110 101 ← d →	

命令群	ニーモニック	オペレーション	S	Z	H	P/V	N	C	バイト	ステート	OPコード 76 543 210
8ビット算術論理演算命令	DEC (IY+d)	(IY+d)←(IY+d)−1	↕	↕	↕	V	1	•	3	23	11 111 101 00 110 101 ← d →
16ビット算術演算命令	ADD HL, ss	HL←HL+ss	•	•	×	•	0	↕	1	11	00 ss1 001 Ⓐ
	ADC HL, ss	HL←HL+ss+CY	↕	↕	×	V	0	↕	2	15	11 101 101 01 ss1 010 Ⓐ
	SBC HL, ss	HL←HL−ss−CY	↕	↕	×	V	1	↕	2	15	11 101 101 01 ss0 010 Ⓐ
	ADD IX, pp	IX←IX+pp	•	•	×	•	0	↕	2	15	11 011 101 00 pp1 001 Ⓒ
	ADD IY, rr	IY←IY+rr	•	•	×	•	0	↕	2	15	11 111 101 00 rr1 001 Ⓓ
	INC ss	ss←ss+1	•	•	•	•	•	•	1	6	00 ss0 011 Ⓐ
	INC IX	IX←IX+1	•	•	•	•	•	•	2	10	11 011 101 00 100 011
	INC IY	IY←IY+1	•	•	•	•	•	•	2	10	11 111 101 00 100 011
	DEC ss	ss←ss−1	•	•	•	•	•	•	1	6	00 ss1 011 Ⓐ
	DEC IX	IX←IX−1	•	•	•	•	•	•	2	10	11 011 101 00 101 011
	DEC IY	IY←IY−1	•	•	•	•	•	•	2	10	11 111 101 00 101 011
アキュムレータ操作命令	DAA	Decimal adjust Acc	↕	↕	↕	P	•	↕	1	4	00 100 111
	CPL	A←Ā	•	•	1	•	1	•	1	4	00 101 111
	NEG	A←Ā+1	↕	↕	↕	V	1	↕	2	8	11 101 101 01 000 100
	CCF	CY←$\overline{CY}$	•	•	×	•	0	↕	1	4	00 111 111
	SCF	CY←1	•	•	0	•	0	1	1	4	00 110 111
CPUコントロール命令	NOP	No operation	•	•	•	•	•	•	1	4	00 000 000
	HALT	CPU halted	•	•	•	•	•	•	1	4	01 110 110
	DI	IFF←0	•	•	•	•	•	•	1	4	11 110 011
	EI	IFF←1	•	•	•	•	•	•	1	4	11 111 011
	IM 0	Set interrupt mode 0	•	•	•	•	•	•	2	8	11 101 101 01 000 110
	IM 1	Set interrupt mode 1	•	•	•	•	•	•	2	8	11 101 101 01 010 110
	IM 2	Set interrupt mode 2	•	•	•	•	•	•	2	8	11 101 101 01 011 110
ローテート・シフト命令	RLCA		•	•	0	•	0	↕	1	4	00 000 111
	RLA		•	•	0	•	0	↕	1	4	00 010 111
	RRCA		•	•	0	•	0	↕	1	4	00 001 111
	RRA		•	•	0	•	0	↕	1	4	00 011 111

命令群	ニーモニック	オペレーション	S	Z	H	P/V	N	C	バイト	ステート	OPコード 76 543 210	
ローテート・シフト命令	RLC r		↕	↕	0	P	0	↕	2	8	11 001 011 00 000 r	Ⓔ
	RLC (HL)		↕	↕	0	P	0	↕	2	15	11 001 011 00 000 110	
	RLC (IX+d)	CY ← 7 ← 0 ←(7) r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	4	23	11 011 101 11 001 011 ← d → 00 000 110	
	RLC (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 000 110	
	RL r		↕	↕	0	P	0	↕	2	8	11 001 011 00 010 r	Ⓔ
	RL (HL)		↕	↕	0	P	0	↕	2	15	11 001 011 00 010 110	
	RL (IX+d)	CY ← 7 ← 0 ← CY r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	4	23	11 011 101 11 001 011 ← d → 00 010 110	
	RL (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 010 110	
	RRC r		↕	↕	0	P	0	↕	2	8	11 001 011 00 001 r	Ⓔ
	RRC (HL)		↕	↕	0	P	0	↕	2	15	11 001 011 00 001 110	
	RRC (IX+d)	(0)→ 7 → 0 → CY r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	4	23	11 011 101 11 001 011 ← d → 00 001 110	
	RRC (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 001 110	
	RR r		↕	↕	0	P	0	↕	2	8	11 001 011 00 011 r	Ⓔ
	RR (HL)		↕	↕	0	P	0	↕	2	15	11 001 011 00 011 110	
	RR (IX+d)	CY → 7 → 0 → CY r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	4	23	11 011 101 11 001 011 ← d → 00 011 110	
	RR (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 011 110	
	SLA r		↕	↕	0	P	0	↕	2	8	11 001 011 00 100 r	Ⓔ
	SLA (HL)		↕	↕	0	P	0	↕	2	15	11 001 011 00 100 110	
	SLA (IX+d)	CY ← 7 ← 0 ← 0 r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	4	23	11 011 101 11 001 011 ← d → 00 100 110	
	SLA (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 100 110	

命令群	ニーモニック	オペレーション	S	Z	H	P/V	N	C	バイト	ステート	OPコード 76 543 210
ローテート・シフト命令	SRA r		↕	↕	0	P	0	↕	2	8	11 001 011 00 101 r Ⓔ
	SRA (HL)		↕	↕	0	P	0	↕	2	15	11 001 011 00 101 110
	SRA (IX+d)	[7 → 0] → [CY] r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	4	23	11 011 101 11 001 011 ← d → 00 101 110
	SRA (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 101 110
	SRL r		↕	↕	0	P	0	↕	2	8	11 001 011 00 111 r Ⓔ
	SRL (HL)		↕	↕	0	P	0	↕	2	15	11 001 011 00 111 110
	SRL (IX+d)	0→[7 → 0] → [CY] r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	4	23	11 011 101 11 001 011 ← d → 00 111 110
	SRL (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 111 110
	RLD	A [7−4│3−0] [7−4│3−0](HL)	↕	↕	0	P	0	·	2	18	11 101 101 01 101 111
	RRD	A [7−4│3−0] [7−4│3−0](HL)	↕	↕	0	P	0	·	2	18	11 101 101 01 100 111
ビット操作命令	BIT b, r	Z←$\overline{r}_b$	×	↕	1	×	0	·	2	8	11 001 011 01 b r ⒺⒻ
	BIT b, (HL)	Z←$\overline{(HL)}_b$	×	↕	1	×	0	·	2	12	11 001 011 01 b 110 Ⓕ
	BIT b, (IX+d)	Z←$\overline{(IX+d)}_b$	×	↕	1	×	0	·	4	20	11 011 101 11 001 011 ← d → 01 b 110 Ⓕ
	BIT b, (IY+d)	Z←$\overline{(IY+d)}_b$	×	↕	1	×	0	·	4	20	11 111 101 11 001 011 ← d → 01 b 110 Ⓕ
	SET b, r	r_b←1	·	·	·	·	·	·	2	8	11 001 011 11 b r ⒺⒻ
	SET b, (HL)	$(HL)_b$←1	·	·	·	·	·	·	2	15	11 001 011 11 b 110 Ⓕ
	SET b, (IX+d)	$(IX+d)_b$←1	·	·	·	·	·	·	4	23	11 011 101 11 001 011 ← d → 11 b 110 Ⓕ
	SET b, (IY+d)	$(IY+d)_b$←1	·	·	·	·	·	·	4	23	11 111 101 11 001 011 ← d → 11 b 110 Ⓕ
	RES b, r	r_b←0	·	·	·	·	·	·	2	8	11 001 011 10 b r ⒺⒻ
	RES b, (HL)	$(HL)_b$←0	·	·	·	·	·	·	2	15	11 001 011 10 b 110 Ⓕ
	RES b, (IX+d)	$(IX+d)_b$←0	·	·	·	·	·	·	4	23	11 011 101 11 001 011 ← d → 10 b 110 Ⓕ
	RES b, (IY+d)	$(IY+d)_b$←0	·	·	·	·	·	·	4	23	11 111 101 11 001 011 ← d → 10 b 110 Ⓕ

命令群	ニーモニック	オペレーション	フラグ S Z H P/V N C	バイト	ステート	OPコード 76 543 210	
ジャンプ・コール・リターン命令	JP nn	PC←nn	• • • • • • •	3	10	11 000 011 ← n → ← n →	
	JP cc, nn	If cc is true PC←nn Otherwise continue	• • • • • • •	3	10	11 cc 010 ← n → ← n →	Ⓖ
	JR e	PC←PC+e	• • • • • • •	2	12	00 011 000 ← e−2 →	
	JR C, e	If C=0 continue If C=1 PC←PC+e	• • • • • • •	2	7 if C=0 12 if C=1	00 111 000 ← e−2 →	
	JR NC, e	If C=1 continue If C=0 PC←PC+e	• • • • • • •	2	7 if C=1 12 if C=0	00 110 000 ← e−2 →	
	JR Z, e	If Z=0 continue If Z=1 PC←PC+e	• • • • • • •	2	7 if Z=0 12 if Z=1	00 101 000 ← e−2 →	
	JR NZ, e	If Z=1 continue If Z=0 PC←PC+e	• • • • • • •	2	7 if Z=1 12 if Z=0	00 100 000 ← e−2 →	
	JP (HL)	PC←HL	• • • • • • •	1	4	11 101 001	
	JP (IX)	PC←IX	• • • • • • •	2	8	11 011 101 11 101 001	
	JP (IY)	PC←IY	• • • • • • •	2	8	11 111 101 11 101 001	
	DJNZ e	B←B−1 if B=0 continue if B≠0 PC←PC+e	• • • • • • •	2	8 if B=0 13 if B≠0	00 010 000 ← e−2 →	
	CALL nn	(SP−1)←PC_H (SP−2)←PC_L PC←nn	• • • • • • •	3	17	11 001 101 ← n → ← n →	
	CALL cc, nn	If cc is false continue otherwise same as CALL nn	• • • • • • •	3	10 if cc is false 17 if cc is true	11 cc 100 ← n → ← n →	Ⓖ
	RET	PC_L←(SP) PC_H←(SP+1)	• • • • • • •	1	10	11 001 001	
	RET cc	If cc is false continue otherwise same as RET	• • • • • • •	1	5 if cc is false 11 if cc is true	11 cc 000	Ⓖ
	RETI	Return from interrupt	• • • • • • •	2	14	11 101 101 01 001 101	
	RETN	Return from non maskable interrupt	• • • • • • •	2	14	11 101 101 01 000 101	
	RST p	(SP−1)←PC_H (SP−2)←PC_L PC_H←0 PC_L←p	• • • • • • •	1	11	11 t 111	Ⓗ
入出力命令	IN A, n	A←(n) $A_{0\sim7}$←n $A_{8\sim15}$←A	• • • • • • •	2	11	11 011 011 ← n →	
	IN r, (C)	r←(C) if r=110 only the flags will be affected $A_{0\sim7}$←C $A_{8\sim15}$←B	↕ ↕ ↕ P 0 •	2	12	11 101 101 01 r 000	Ⓔ
	INI	(HL)←(C) B←B−1 HL←HL+1 $A_{0\sim7}$←C $A_{8\sim15}$←B	× ↕③ × × 1 •	2	16	11 101 101 10 100 010	
	INIR	(HL)←(C) B←B−1 HL←HL+1 until B=0 $A_{0\sim7}$←C $A_{8\sim15}$←B	× 1 × × 1 •	2	21 if B≠0 16 if B=0	11 101 101 10 110 010	
	IND	(HL)←(C) B←B−1 HL←HL−1 $A_{0\sim7}$←C $A_{8\sim15}$←B	× ↕③ × × 1 •	2	16	11 101 101 10 101 010	

命令群	ニーモニック	オペレーション	フラグ S Z H P/V N C	バイト	ステート	OPコード 76 543 210
入出力命令	INDR	(HL)←(C) B←B−1 HL←HL−1 until B=0 $A_{0\sim7}$←C $A_{8\sim15}$←B	× 1 × × 1 •	2	21 if B≠0 16 if B=0	11 101 101 10 111 010
	OUT n, A	(n)←A $A_{0\sim7}$←n $A_{8\sim15}$←A	• • • • • • •	2	11	11 010 011 ← n →
	OUT (C), r	(C)←r $A_{0\sim7}$←C $A_{8\sim15}$←B	• • • • • • •	2	12	11 101 101 01 r 001 Ⓔ
	OUTI	(C)←(HL) B←B−1 HL←HL+1 $A_{0\sim7}$←C $A_{8\sim15}$←B	× ↕③ × × 1 •	2	16	11 101 101 10 100 011
	OTIR	(C)←(HL) B←B−1 HL←HL+1 until B=0 $A_{0\sim7}$←C $A_{8\sim15}$←B	× 1 × × 1 •	2	21 if B≠0 16 if B=0	11 101 101 10 110 011
	OUTD	(C)←(HL) B←B−1 HL←HL−1 $A_{0\sim7}$←C $A_{8\sim15}$←B	× ↕③ × × 1 •	2	16	11 101 101 10 101 011
	OTDR	(C)←(HL) B←B−1 HL←HL−1 until B=0 $A_{0\sim7}$←C $A_{8\sim15}$←B	× 1 × × 1 •	2	21 if B≠0 16 if B=0	11 101 101 10 111 011

Ⓐ		Ⓑ		Ⓒ		Ⓓ		Ⓔ		Ⓕ		Ⓖ				Ⓗ	
Reg	ss dd	Reg	qq	Reg	pp	Reg	rr	Reg	r, r'	Bit	b	cc	Condition		Flag	P	t
BC	00	BC	00	BC	00	BC	00	B	000	0	000	000	NZ	Non Zero	Z	00H	000
DE	01	DE	01	DE	01	DE	01	C	001	1	001	001	Z	Zero	Z	08H	001
HL	10	HL	10	IX	10	IY	10	D	010	2	010	010	NC	Non Carry	C	10H	010
SP	11	AF	11	SP	11	SP	11	E	011	3	011	011	C	Carry	C	18H	011
								H	100	4	100	100	PO	Parity Odd	P/V	20H	100
								L	101	5	101	101	PE	Parity Even	P/V	28H	101
								A	111	6	110	110	P	Sign Positive	S	30H	110
										7	111	111	M	Sign Negative	S	38H	111

d, n ：8ビット・イミーディエト・データ
e ：相対アドレシングの変位置
e−2：eの実効変位置

フラグ

•：影響受けない　↕：演算結果に従った影響を受ける　① BC−1=0 ならば P/V=0, その他 P/V=1
0：リセット　P："1" 偶数パリティ, "0" 奇数パリティ　② A=(HL) ならば Z=1, その他 Z=0
1：セット　V："1"オーバフロー有り, "0"オーバフロー無し　③ B−1=0 ならば Z=1, その他 Z=0
×：不定　IFF：P/Vフラグ←(IFF)

Z80 ニーモニック⟷マシン語対照表

8ビット・ロード

＼ ×	I	R	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX+d)	(IY+d)	(nn)	n
LD A, ×	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n
LD B, ×			47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n
LD C, ×			4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n
LD D, ×			57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n
LD E, ×			5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n
LD H, ×			67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n
LD L, ×			6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n
LD (HL), ×			77	70	71	72	73	74	75							36 n
LD (BC), ×			02													
LD (DE), ×			12													
LD (IX+d), ×			DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n
LD (IY+d), ×			FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n
LD (nn), ×			32 n n													
LD I, ×			ED 47													
LD R, ×			ED 4F													

16ビット・ロード

＼ ×	AF	BC	DE	HL	SP	IX	IY	nn	(nn)
LD AF, ×									
LD BC, ×								01 n n	ED 4B n n
LD DE, ×								11 n n	ED 5B n n
LD HL, ×								21 n n	2A n n
LD SP, ×				F9		DD F9	FD F9	31 n n	ED 7B n n
LD IX, ×								DD 21 n n	DD 2A n n
LD IY, ×								FD 21 n n	FD 2A n n
LD (nn), ×		ED 43 n n	ED 53 n n	22 n n	ED 73 n n	DD 22 n n	FD 22 n n		
PUSH ×	F5	C5	D5	E5		DD E5	FD E5		
POP ×	F1	C1	D1	E1		DD E1	FD E1		

ブロック転送

LDI	ED A0
LDIR	ED B0
LDD	ED A8
LDDR	ED B8

ブロック・サーチ

CPI	ED A1
CPIR	ED B1
CPD	ED A9
CPDR	ED B9

ローテート，シフト

×	A	B	C	D	E	H	L	(HL)	(IX +d)	(IY +d)
RLC ×	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
RRC ×	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
RL ×	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
RR ×	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
SLA ×	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 26	FD CB d 26
SRA ×	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
SRL ×	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
RLD								ED 6F		
RRD								ED 67		

	A
RLCA	07
RRCA	0F
RLA	17
RRA	1F

ジャンプ，コール，リターン

×	UN COND	C	NC	Z	NZ	PE	PO	M	P	
JP ×, nn	C3 n n	DA n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
JR ×, e	18 e−2	38 e−2	30 e−2	28 e−2	20 e−2					
JP (HL)	E9									
JP (IX)	DD E9									
JP (IY)	FD E9									
CALL ×, nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
DJNZ e										10 e−2
RET ×	C9	D8	D0	C8	C0	E8	E0	F8	F0	
RETI	ED 4D									
RETN	ED 45									

リスタート

RST 00H	C7
RST 08H	CF
RST 10H	D7
RST 18H	DF
RST 20H	E7
RST 28H	EF
RST 30H	F7
RST 38H	FF

ビット操作

＼ ×	A	B	C	D	E	H	L	(HL)	(IX +d)	(IY +d)
BIT 0, ×	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
BIT 1, ×	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
BIT 2, ×	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
BIT 3, ×	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
BIT 4, ×	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
BIT 5, ×	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
BIT 6, ×	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
BIT 7, ×	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
RES 0, ×	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
RES 1, ×	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
RES 2, ×	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
RES 3, ×	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
RES 4, ×	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
RES 5, ×	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
RES 6, ×	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
RES 7, ×	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET 0, ×	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
SET 1, ×	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
SET 2, ×	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
SET 3, ×	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
SET 4, ×	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
SET 5, ×	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
SET 6, ×	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
SET 7, ×	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

入　力

IN A, n	DB n
IN A, (C)	ED 78
IN B, (C)	ED 40
IN C, (C)	ED 48
IN D, (C)	ED 50
IN E, (C)	ED 58
IN H, (C)	ED 60
IN L, (C)	ED 68
INI	ED A2
INIR	ED B2
IND	ED AA
INDR	ED BA

出　力

OUT n, A	D3 n
OUT (C), A	ED 79
OUT (C), B	ED 41
OUT (C), C	ED 49
OUT (C), D	ED 51
OUT (C), E	ED 59
OUT (C), H	ED 61
OUT (C), L	ED 69
OUTI	ED A3
OTIR	ED B3
OUTD	ED AB
OTDR	ED BB

ＰＣ－8801　キャラクタ・コード表

上位４ビット→　（列）　下位４ビット↓　（行）

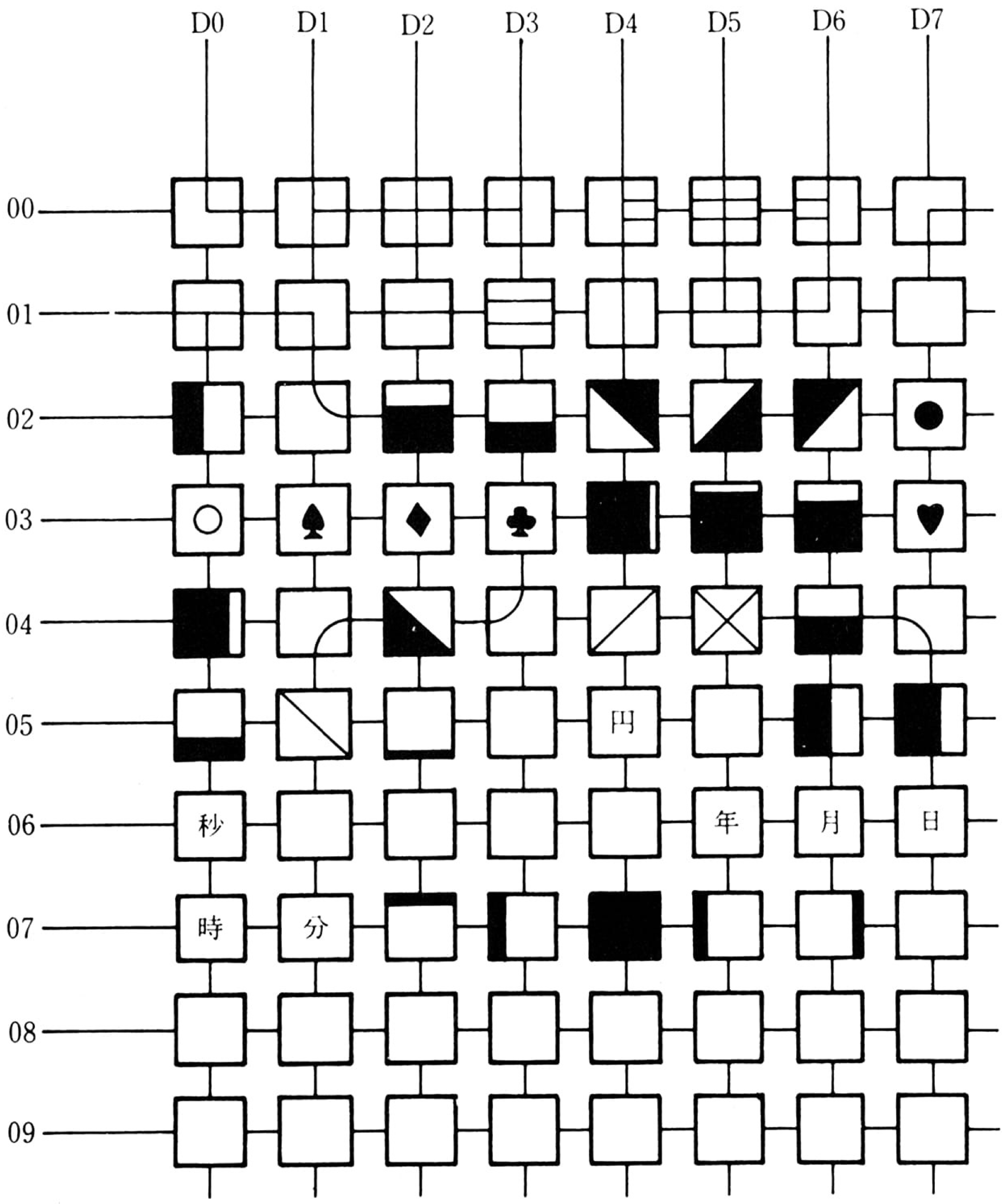
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		D_E		0	@	P		p				ー	タ	ミ		
1	S_H	D_1	!	1	A	Q	a	q			。	ア	チ	ム		円
2	S_X	D_2	"	2	B	R	b	r			「	イ	ツ	メ		年
3	E_X	D_3	#	3	C	S	c	s			」	ウ	テ	モ		月
4	E_T	D_4	$	4	D	T	d	t			、	エ	ト	ヤ		日
5	E_Q	N_K	%	5	E	U	e	u			・	オ	ナ	ユ		時
6	A_K	S_N	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		分
7	B_L	E_B	'	7	G	W	g	w			ァ	キ	ヌ	ラ		秒
8	B_S	C_N	(	8	H	X	h	x			ィ	ク	ネ	リ	♠	
9	H_T	E_M	)	9	I	Y	i	y			ゥ	ケ	ノ	ル	♥	
A	L_F	S_B	*	:	J	Z	j	z			ェ	コ	ハ	レ	♦	
B	H_M	E_C	+	;	K	[	k	{			ォ	サ	ヒ	ロ	♣	
C	C_L	→	,	<	L	¥	l	|			ャ	シ	フ	ワ	●	
D	C_R	←	-	=	M	]	m	}			ュ	ス	ヘ	ン	○	
E	S_O	↑	.	>	N	∧	n	~			ョ	セ	ホ	゛		
F	S_I	↓	/	?	O	—	o				ッ	ソ	マ	゜		

キーボード・マップ

（データ・バス）

（アドレス・バス）

	D0	D1	D2	D3	D4	D5	D6	D7
00	0	1	2	3	4	5	6	7
01	8	9	*	+	=	,	.	RETURN
02	@ ゛	A チ	B コ	C ソ	D シ	E イ	F ハ	G キ
03	H ク	I ニ	J マ	K ノ	L リ	M モ	N ミ	O ラ
04	P セ	Q タ	R ス	S ト	T カ	U ナ	V ヒ	W テ
05	X サ	Y ン	Z ッ ツ	[。 「	¥ _	] ム 」	^ ヘ	－ ＝ ホ
06	0 ヲ ワ	1 ! ヌ	2 " フ	3 # ァ ア	4 $ ゥ ウ	5 % ェ エ	6 & ォ オ	7 ' ャ ヤ
07	8 (ュ ユ	9) ョ ヨ	: * ケ	; + レ	, < 、 ネ	. > 。 ル	/ ? ・ メ	－ ロ
08	HOME CLR	↑	→	INS DEL	GRAPH	カナ	SHIFT	CTRL
09	STOP	f・1	f・2	f・3	f・4	f・5	SPACE	ESC
0A	HTAB	↓	←	HELP	COPY	—	／	CAPS LOCK
0B	ROLL UP	ROLL DOWN						


D0
D1
D2
D3
D4
D5
D6
D7
00
01
02
03
04
05
06
07
08
09
円
秒
年
月
日
時
分

索引

A

B

C

D

E

G

H

I

S

U

V

W

X

PC-8801＋mkII
マシン語プログラミング入門

著　者　森　巧尚　
発行者　田村正隆

発行所　株式会社　ナツメ社
東京都千代田区神田神保町1－52（〒101）
電話　03（291）1257（代表）
振替　東京3－58661
印　刷　ラン印刷社
製　本　三修紙工

ISBN4-0404-5　Printed in Japan

PC-9801 プログラミング入門

野々山隆幸著　1500円

●ズブの素人にも理解できる恰好の入門書です．漢字の使い方も解説してあります．

【内容】◆デモプログラムのロード　◆関数計算もできます　◆N88-BASICのスタート方法　◆システムディスクのバックアップ　◆プログラムの作成　◆プログラムのセーブ　◆経営分析プログラム　◆データインプットのプログラム　◆設備投資計画プログラム　◆棒グラフの作成　◆エンド部分とサブルーチンのプログラム　◆利益計画プログラム　◆損益分岐点図表プログラム　◆予算実績差異分析プログラム　◆PC-8001用プログラムを使うには

PC-9801 ビジネス・プログラミング

布田 治著　1500円

●ビジネスへの適用の手順を販売管理を例にとってやさしく解説してあります．ディスクベースですので，ディスクをお持ちの読者向けです．ビジネス・ソフトの基本は，この本にすべて含まれています．

【内容】◆問題解決の定石　◆ディスクシステム「ひらけゴマ」　◆ディスクに何か書き込もう　◆入力プログラムの作成　◆漢字で見出しを作ろう　◆項目の印刷方法アラカルト　◆どう並べかえるか　◆分類プログラムの実行　◆マッチングプログラムの実行　◆印刷文字のいろいろ　◆ランダムファイルってなあに　◆書いたものは読めなきゃウソ　◆更新の「三種の神器」　◆漢字名称はクセモノ　◆インデックスがなきゃランダムじゃないよ　◆分析表の作成　◆いまやグラフィック時代　◆顔は円グラフ　◆最後に笑うのはCHAINくん

PC-9801 アセンブリ言語プログラミング入門

川村 清著　1600円

●16ビットマイコン８０８６のマシン語入門です．短いサンプルプログラムを数多くあげ，実行結果・入力方法などをディスプレイ表示にしてありますので，大変理解しやすくなっています．

【内容】◆ＰＣ-9801のOS　◆メモリのセグメンテーション　◆機械語モニタの起動方法　◆メモリの内容をコピーするプログラム　◆メッセージを表示するプログラム　◆16ビットの加減算を行なうプログラム　◆16ビットデータを10倍するプログラム　◆サブルーチンの使い方　◆サブルーチン・コールとレジスタ退避　◆分岐のあるプログラム　◆くり返し処理のあるプログラム　◆簡単なデモンストレーション　◆８０８６のインストラクション活用表

PC-8801 プログラミング入門

野々山隆幸著　　1400円

●ズブの素人にも理解できる恰好の入門書です．漢字の使い方も解説してあります．

【内容】 ◆デモプログラムのロード ◆関数計算もできます ◆N88-BASICのスタート方法 ◆システムディスクのバックアップ ◆プログラムの作成 ◆プログラムのセーブ ◆経営分析プログラム ◆データインプットのプログラム ◆設備投資計画プログラム ◆棒グラフの作成 ◆エンド部分とサブルーチンのプログラム ◆利益計画プログラム ◆損益分岐点図表プログラム ◆予算実績差異分析プログラム ◆PC-8001用プログラムを使うには

PC-8801 ビジネス・プログラミング

布田 治著　　1500円

●ビジネスへの適用の手順を販売管理を例にとってやさしく解説してあります．ディスクベースですので，ディスクをお持ちの読者向けです．ビジネス・ソフトの基本は，この本にすべて含まれています．

【内容】 ◆問題解決の定石 ◆ディスクシステム「ひらけゴマ」 ◆ディスクに何か書き込もう ◆入力プログラムの作成 ◆漢字で見出しを作ろう ◆項目の印刷方法アラカルト ◆どう並べかえるか ◆分類プログラムの実行 ◆マッチングプログラムの実行 ◆印刷文字のいろいろ ◆ランダムファイルってなあに ◆書いたものは読めなきゃウソ ◆更新の「三種の神器」 ◆漢字名称はクセモノ ◆インデックスがなきゃランダムじゃないよ ◆分析表の作成 ◆いまやグラフィック時代 ◆顔は円グラフ ◆最後に笑うのはCHAINくん

PC-8801 アセンブリ言語プログラミング入門

脇 英世著　　1600円

●内蔵アセンブラを使ったマシン語入門です．具体的な短いプログラムをサンプルに解説していますので，大変わかりやすくなっています．数値のソート，文字のソートまで扱っています．

【内容】 ◆BASICとアセンブリ言語 ◆予備知識はいりません ◆モニタプログラムの動かし方 ◆1と0をひっくり返すこと ◆大きい数をさがすこと ◆16ビットの足し算 ◆引き算はこうします ◆和を計算する ◆ASCII符号とはなんだろう ◆同じ文字列かどうかしらべる ◆精度の高い足し算 ◆簡単な並べかえ ◆キーワードとジャンプテーブルを使う ◆PC-8801のBASICを探検する ◆BASICと機械語のプログラムをつなげる ◆16ビットのデータを並べかえる ◆機械語プログラムの呼び出し方

PC-8801mkⅡマシン語活用マニュアル

塚越一雄著　1800円

【内容】◆MON処理ルーチンの解析　◆マシン語モニタの解析　◆GコマンドとROMセレクトの秘密　◆ブレーク・ポイント処理のしくみ　◆技術情報発見のルーツ　◆ユーザー・スタック・エリアを探る　◆フック・アドレスの活用　◆N88-BASIC・ROM内システム・サブルーチン　◆サブROM内システム・サブルーチン　◆CLEAR文自動設定サブルーチン　◆画面クリアとカーソル制御　◆テキスト画面のコントロール　◆リアルタイム・キー入力　◆キー先行入力　◆自動オペレーションの設定　◆カセット・レコーダの制御　◆プリンタの制御　◆ディスクへの入出力　◆カラー・グラフィック

PC-8801+mkⅡマシン語活用入門

塚本浩二著　1200円

【内容】◆マシン語とは何か　◆マシン語は難しくない　◆ニーモニックは人間的だ　◆マシン語マスターへの近道　◆内部ルーチンを使え！　◆マシン語の基礎知識　◆レジスタとは　◆フラグとは何か　◆LD命令　◆JP命令とCP命令　◆CALL命令とRET命令　◆アセンブラのための命令　◆マシン語からBASICへ　◆マシン語からモニタへ　◆WIDTHをマシン語で　◆画面設定内部ルーチンあれこれ　◆1文字表示内部ルーチン　◆マシン語で文字列をPRINT

PC-8801漢字ワードプロセッサ入門

安立隆一著　1300円

●本書はPC-8801mkIIにも使えます。

【内容】◆ワードプロセッサとは？　◆どんな機能が必要か？　◆おおもとのプログラム　◆バッファクリアコマンド　◆タイムコマンド　◆データディスプレイコマンド　◆ランダムデータライトコマンド　◆データメイドコマンド　◆データメイドコマンドの終了　◆データプリントコマンド　◆データスピークコマンド　◆ワープロの使い方　◆時間のセット　◆記憶エリアの初期化　◆デタラメな文章を作る　◆データの表示　◆文章の作成

PC-8801mkⅡプログラミング入門

編集部編著　1400円

●わかりやすいと評判の入門書です．拡張機能のサウンド，タートルグラフィックについても，その使い方をやさしく解説してあります．

【内容】 ◆キー操作を覚えよう　◆画面に文字を表示する　◆画面をカラーに　◆まず，小さなプログラムから　◆プログラムのセーブとロード　◆表を作ってみよう　◆画面のレイアウト　◆ボールが何に当たった？　◆ボール表示とブロック消去　◆メニュー画面　◆もっとチャレンジしたければ　◆プログラムの完成　◆DMプログラム　◆メインルーチン　◆データサーチ　◆宛て名の表示　◆サウンド機能を活かそう　◆ブロックくずしに装飾音を　◆タートルグラフィックの作い方

PC-8801mkⅡグラフィックス入門

脇　英世著　1600円

【内容】 ◆グラフィック画面の表示を決めること　◆座標系とは何だろう　◆カラーパレットとは何だろう　◆WINDOWとVIEWのやさしい使い方　◆円はどうやって描くのだろう　◆SCREEN文を自由に使いこなそう　◆エンタープライズ号をDATA文で描く　◆特別な入力機器を使わずに絵を描く　◆ゴールデンゲートブリッジを作る　◆タイルパターンを作るためのツール　◆タイル文によらない色の作り方　◆物体を回転させてみよう　◆物体を拡大・縮小する　◆3次元空間における平行移動　◆ワイヤーフレームモデルを作る　◆地球に穴をあける　◆エンタープライズ号の完成

PC-8801mkⅡアニメーション・グラフィックス

安立雅子著　1200円

●本書はPC-8801にも使えます。

【内容】 ◆アニメを描こう　◆アニメ・ドローイングの考え方　◆いよいよプログラム　◆バッファクリア　◆データ作成　◆色ぬりコマンド　◆アニメ作成　◆座標データ入力　◆コードデータ入力　◆データ表示　◆プログラム作成　◆アニメを作ろう　◆イラスト作画プログラム　◆プログラムの製作　◆イラストを描く

PC-8801mKⅡデータファイル入門(1)

編集部編著　　1300円

【内容】◆操作する前に　◆システムディスクのコピーを作る　◆プログラムのセーブとロード　◆ディスクの管理　◆ファイル名の変更と削除　◆ディスクの構造　◆ディスクにデータを書き込む　◆ディスクのデータを読む　◆データを追加する　◆データ修正ルーチン　◆メニュープログラムを作ろう　◆電話帳ファイルの作成　◆登録・追加プログラムの実行　◆データの検索　◆データを印刷する　◆ランダムファイルについて　◆メニュープログラムとデータの書き込み　◆宛て名の印刷　◆ディスクの有効利用　◆マスターメニュープログラム

PC-8801mkⅡデータファイル入門(2)

編集部編著　　1200円

【内容】◆ファイル内でのデータ設計　◆データの書き込み　◆書き込みデータの確認　◆データの追加　◆データの修正　◆宛て名の印刷　◆コード入力による宛て名印刷　◆メニュープログラム　◆ファイルに，より資料性を　◆プログラムの動作テスト　◆データ更改のプログラム　◆６つの印刷メニュー　◆漢字をプリンタで出力　◆文書のデータファイルを作る　◆漢字で通信文を　◆マスターメニュープログラム

PC-8801mkⅡデータファイル活用法

編集部編著　　1800円

【内容】◆システムフロー　◆データのフォーマットを決める　◆初期設定からファイルオープンまで　◆プログラムの動作テスト　◆データのソート　◆商品名のリストアウト　◆プログラムチェックとデータ書き込み　◆サンプルデータの入力　◆データの並べ換え　◆得意先データのリストアウト　◆ファイル名の決定とサブルーチン　◆データをディスクにセット　◆エンディング処理　◆月末処理ガイドプログラム　◆入出庫一覧表の印刷　◆得意先別売り上げ計算　◆任意のプログラムのロード　◆システムの実行　◆ＦＡＴをのぞく　◆ＩＤセクタをのぞく　◆ＫＩＬＬしてもデータは死なない！　◆データファイルの転送

PC-8801+mkⅡ
マシン語プログラミング入門

発行――1984年 9 月20日
著者――森巧尚
発行者――田村正隆
発行所――株式会社ナツメ社
郵便番号101
東京都千代田区神田神保町1-52
電話〈03〉291-1257
振替 東京3-58661
〈落丁・乱丁本はお取り替えします〉

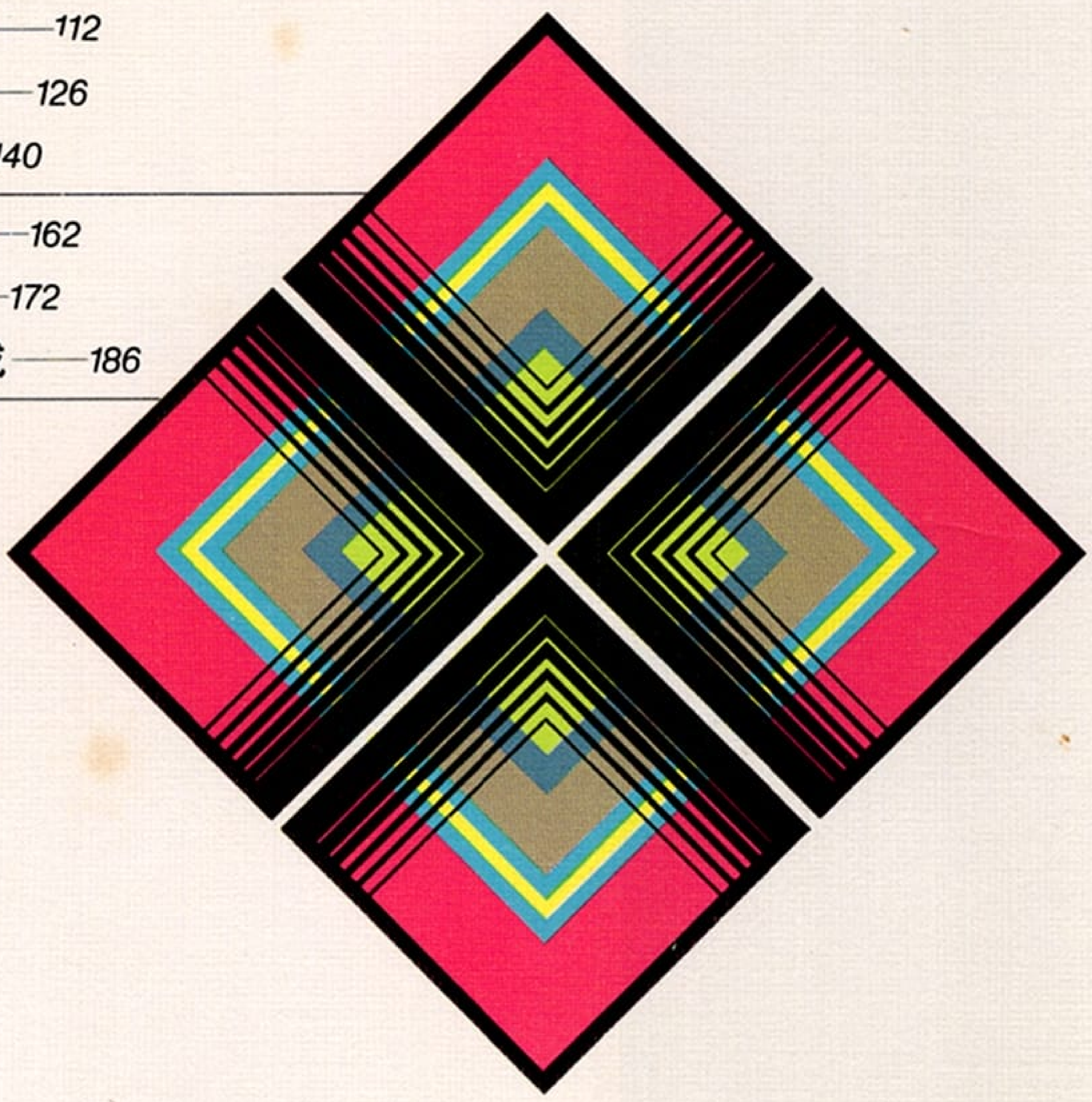
定価――1500円

PC-8801+mkII

マシン語プログラミング入門

[目次より]

ナツメ社
定価＝1500円



ISBN-4-8163-0404-5 C2054 ¥1500E